Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Омский государственный технический университет»

На правах рукописи

Панков Денис Анатольевич

# СПОСОБЫ И АЛГОРИТМЫ ТЕСТИРОВАНИЯ ПРОГРАММНО-АППАРАТНЫХ КОМПЛЕКСОВ НА ОСНОВЕ ИМИТАЦИИ НЕИСПРАВНОСТЕЙ

Специальность: 05.13.01 – «Системный анализ,
управление и обработка информации»

ДИССЕРТАЦИЯ
на соискание ученой степени
кандидата технических наук

Научный руководитель
доктор технических наук, доцент
Денисова Людмила Альбертовна

Омск – 2021

# ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

**Актуальность темы исследования.** В настоящее время одной из важных задач современной отечественной промышленности является разработка сложных устройств с уникальными потребительскими свойствами, способных выполнять свои функции с высоким уровнем устойчивости к отказам и сбоям. При этом согласно Стратегии развития информационных технологий Российской Федерации на период до 2025 года для аппаратно-программных комплексов с большим удельным весом программной части внимание уделяется исследованию методов, ведущих к сокращению сроков разработки образцов и повышению их отказоустойчивости.

Совершенствование методов и средств повышения отказоустойчивости для микропроцессорных устройств связано с проблемой эффективности применения известных стандартов и методик для малосерийных устройств информационных комплексов. Кроме того, для регламентируемой в стандартах процедуры тестирования цифровых устройств с помощью имитации неисправностей не определено конкретных алгоритмов и необходимого комплекса технических средств.

Теоретическое и практическое решение проблемы обеспечения повышения устойчивости к отказам и сбоям для проектируемых микропроцессорных устройств может быть достигнуто путем синтеза способов экспериментальных исследований по имитации неисправностей с помощью искажения входных данных и предобработки статистической информации для программного обеспечения (ПО), загруженного в аппаратную среду микропроцессорного устройства, а также анализа реакции на внесенные неисправности и определения достаточности объема испытаний с помощью нечеткого логического вывода.

**Степень разработанности темы исследования.** Современные стандарты разработки и проектирования технических программно-аппаратных устройств, основанные на методике FMEA, находятся в системном кризисе в связи с постоянным ростом сложности систем управления информационных комплексов и условий их

применения, с одной стороны, и в связи с проявлением свойства эмерджентности этих систем, которое не учитывается на этапе проектирования. Исследования по получению математических моделей возникновения отказов и сбоев для построения систем управления проводились ведущими в этом направлении организациями, такими как Институт проблем управления им. В.А. Трапезникова Российской академии наук (ИПУ РАН), АО «ИСС» имени академика М. Ф. Решетнёва», Институт комплексной автоматизации (ЦНИИКА), ЦНИИ 22 МО РФ, *NASA Jet Propulsion Laboratory* (*JPL*), *Massachusetts Institute of Technology* (*MIT*), коммерческие фирмы «*Exic*» и «*Simic*».

Значительный вклад в повышение устойчивости к отказам и сбоям сложных программно-аппаратных устройств внесли отечественные и зарубежные ученые Пархоменко П.П., Черкесов Г.Н., Ушаков И.В., Avizienis A., Randell B., Herrin S., Katsuki D., Globl W. и др. Вопросами проектирования средств имитации неисправностей занимаются ученые Лучинин В.В., Arlant J., Elks C., Rannels D., Williams R., Gaisler J., Natella R.

Вопросам повышения качества и надежности ПО посвящены работы ученых Липаева В.В., Ковалева И.В., Boehm B. W., Meyer J., Lyu M. R., Levendel Y., Shooman M. L., Tai A., Xie M, Zhou L. Исследованиям свойств программного обеспечения и безопасности программ посвящены работы Berman O., Choi J. G., Epstein D., She D., Pei K.

Существует список так и не решённых в теории проблем надежности, который был опубликован на Международной конференции *MMR*-2000 в докладе «Надежность: прошлое, настоящее, будущее». На первом месте в этом списке находится проблема надёжности программного обеспечения, на третьем месте – вопросы надежности уникальных высокоответственных систем. Проблемы надёжности глобальных территориальных систем и телекоммуникационных сетей занимают четвёртое и пятое место. К решению этих проблем есть только подходы.

Теория надёжности сформировалась как часть прикладной математики и разработала различные методы решения проблем надёжности, развитые в математические модели, которые все еще не достигли уровня, пригодного для

применения на практике (проблема отсутствия необходимых исходных данных). На практике согласно рекомендации ЦНИИ 22 МО РФ по надежностно-ориентированному проектированию и изготовлению электронных устройств начиная с 1995 года декларировалась обязательность применения методики FMEA в процессе разработки радиоэлектронной аппаратуры (РЭА) со сложным программным обеспечением на всех этапах создания устройств.

Процедура анализа видов, причин и последствий отказов РЭА является обязательной составной частью процесса проектирования и отработки устройств, начиная с разработки эскизного проекта до испытаний опытных образцов. Необходимость проведения FMEA в процессе разработки и проектирования была определена в качестве дополнительного этапа (по отношению к стандарту менеджмента качества ИСО 9001) и составляет требования для поставщиков в рамках корпоративной системы.

На основе проведенного анализа сделан вывод о целесообразности разработки технических решений для практического применения методики FMEA к микропроцессорным устройствам в виде разработанных алгоритмов, методик и устройств.

**Основная идея работы** состоит в использовании предобработанных статистических данных о функционировании проверяемого устройства в алгоритмах имитации и выявления ошибок ПО и аппаратных отказов.

**Целью диссертационной работы** является повышение устойчивости к отказам и сбоям микропроцессорных устройств с помощью технических и программных средств тестирования на основе имитации неисправностей информационного комплекса. Для достижения цели в работе поставлены и решены следующие **задачи**:

1. Анализ проблемы устойчивости к отказам и сбоям программно-аппаратных комплексов и подходов к обеспечению их работоспособности в процессе проектирования.

2. Разработка алгоритма анализа проблемных ситуаций, возникающих при работе микропроцессорного устройства, на основе обнаружения неисправностей с помощью искажения входных данных.

3. Разработка алгоритма поиска неисправностей на основе искажения входных данных основных программных функций с использованием предобработанных статистических показателей работы устройства.

4. Создание технических средств имитации неисправностей и алгоритма диагностики для микропроцессорных устройств в составе информационного комплекса.

5. Разработка метода и критерия определения объема тестовых испытаний на основе статистически обработанных экспериментальных данных о работе устройства. Создание математической модели для обоснования критерия и алгоритма принятия решений о достаточности объема испытаний.

6. Реализация и внедрение полученных теоретических результатов в виде методик, моделей, алгоритмов и программ, используемых при анализе и синтезе микропроцессорных устройств для информационного комплекса.

**Научная новизна.** В процессе исследований получены следующие новые научные результаты.

1. Разработан алгоритм анализа проблемных ситуаций, возникающих при работе автоматизированной радиостанции (патент № 2696977), интегрируемой в действующую инфраструктуру цифровых сетей с множественным доступом. Алгоритм реализует тестирование функций радиостанции с помощью техники фаззинга (случайного внесения искажений в данные). Это позволяет выполнять автоматическую диагностику технического состояния радиостанции и устранять системные ошибки, возникающие при загрузке пользовательских программ в реальную аппаратную среду устройства (проявление свойства эмерджентности).

2. Предложена модификация алгоритма фаззинга, реализующего эволюционную стратегию трансформации входных данных проверяемого устройства. Алгоритм рассматривает набор программных функций, представленный в виде вершин графа переходов. Стратегия фаззинга

предусматривает на каждом этапе изменения данных (перестановка битов и байтов, арифметические операции), анализ реакции программы на внесенные искажения. Модификация алгоритма заключается в применении в качестве входных данных устройства предобработанных массивов наиболее часто выполняемых функций, что позволяет оценить корректность работы для основных уязвимостей ПО и уменьшить временные затраты на тестирование (приблизительно в три раза по сравнению с классическими методами тестирования ПО).

3. Предложено устройство имитации неисправностей (патент № 2697629) в программно-аппаратных системах для проверки работы микропроцессорного устройства и разработан экспертный алгоритм, предназначенный для выявления наиболее уязвимых мест программы и основанный на анализе признаков отказов и сбоев в регистрах ОЗУ микропроцессорного устройства. Алгоритм позволяет исказить входные данные и обнаружить изменения характеристик работы проверяемого устройства, обусловленных изменением порядка и времени выполнения программных функций, что свидетельствует о наличии отказов или сбоев.

4. Разработан метод и предложен критерий определения объема тестовых испытаний микропроцессорных устройств. Создана математическая модель для обоснования критерия, учитывающего предварительно полученные характеристики устройства: число аппаратных отказов, вероятность ошибок при выполнении программных функций, а также корреляцию между временем внесения ошибок во входные данные и временем выявления неисправностей. Метод, основанный на предварительной статистической обработке экспериментальных данных о выполняемых функциях в совокупности с алгоритмом принятия решения на базе нечеткого логического вывода, позволяет оценить достаточность объема испытаний.

5. Предложен алгоритм диагностики программно-аппаратного комплекса, который позволяет оптимизировать скорость передачи данных путем поиска кратчайшего пути в совокупности с ранжированием интерфейсов приема/передачи при наличии большого числа интерфейсов (узлов). Подтверждена эффективность

алгоритма в условиях ограничения на время обмена информацией между узлами (время уменьшается примерно в два раза).

**Практическая значимость работы** заключается в разработке:

– методики и алгоритма анализа проблемных ситуаций автоматизированной радиостанции, интегрируемой в действующую инфраструктуру цифровых сетей с множественным доступом, который реализует тестирование функций радиостанции с помощью техники фаззинга (случайного внесения искажений в данные) и выявляет системные дефекты на основе обработки информации, полученной с помощью интерфейса отладки JTAG;

– программного комплекса для автоматизации проведения испытаний микропроцессорных устройств (в комплексной инфраструктуре тестирования), включающего программу внесения ошибок во входные данные (свидетельство о регистрации программы для ЭВМ № 2019661730), а также программу имитации системных событий (возникновения условий приема и передачи данных) и сбора информации о реакции устройства на внесенные искажения с контрольно-измерительных приборов (свидетельство о регистрации программы для ЭВМ № 2019664742);

– макета имитатора неисправностей для проверки работы микропроцессорных устройств, позволяющего выявить основные уязвимости ПО устройства путем сравнения параметров его состояния со штатными значениями, результатом которого является выявление точек возникновения отказов и сбоев проверяемого устройства.

**Внедрение результатов исследований**. Разработанные методики, алгоритмы и программно-алгоритмические средства для имитации и выявления неисправностей проверяемых устройств внедрены в АО ОНИИП и использованы в рамках выполнения ОКР «Бумеранг», что позволило повысить устойчивость устройств разрабатываемых программно-аппаратных комплексов к отказам и сбоям.

**Объектом** исследования являются способы исследования устойчивости к отказам и сбоям микропроцессорных устройств с программным обеспечением.

**Предметом** исследования являются методы, алгоритмы и математические модели, позволяющие обеспечить работоспособность микропроцессорного устройства.

**Методология исследования** базируется на основах системного анализа, методах теории вероятностей и математической статистике; теории принятия решений, включая методы теории важности критериев; методах тестирования и разработки программно-аппаратных систем, методике имитации неисправностей и технике фаззинга.

**Основные результаты, полученные автором и выносимые на защиту:**

1. Способ и алгоритм анализа проблемных ситуаций автоматизированной радиостанции, позволяющие выполнять автоматическую диагностику технического состояния радиостанции в штатных режимах и устранять системные ошибки с помощью анализа реакции устройства на имитируемые неисправности. В отличие от известных, данный способ, основанный на технике фаззинга, производит случайное внесение искажений во входные данные во всех режимах работы радиостанции и обнаруживает системные ошибки.

2. Способ и алгоритм имитации неисправностей, которые в отличие от известных используют предобработку показателей работы устройства (частота использование функций, ассемблерных команд для режимов функционирования), тем самым позволяя уменьшить временные затраты на проведение испытаний за счет направленного фаззинга для наиболее уязвимых мест программы.

3. Метод и модель определения объема тестирования микропроцессорных устройств, а также критерий завершения испытаний, основанный на экспериментальных показателях работы устройства. В отличие от известных методов, решение о завершении испытаний принимается с помощью нечеткого логического вывода на основе статистически обработанных данных, характеризующих выполнение программных функций, что повышает эффективность выявления дефектов ПО и аппаратных отказов.

4. Алгоритм обмена в системе связи, повышающий скорость передачи данных путем поиска кратчайшего пути между узлами (объектами связи)

программно-аппаратного комплекса. В отличие от известных, алгоритм выполняет диагностику состояния узлов и ранжирование интерфейсов приема/передачи в условиях динамического измерения доступности узлов, что повышает эффективность обмена при наличии большого числа узлов в комплексе.

**Соответствие паспорту специальности.** Диссертация соответствует областям исследований: п. 11 «Методы и алгоритмы прогнозирования и оценки эффективности, качества и надежности сложных систем», п. 13 «Методы получения, анализа и обработки экспертной информации» научной специальности 05.13.01 «Системный анализ, управление и обработка информации».

**Достоверность полученных результатов.**

Обоснованность и достоверность теоретических результатов, положений и выводов, полученных в диссертационной работе, базируются на использовании апробированных научных положений и методов исследования, корректном применении математического аппарата, согласованности новых результатов с известными теоретическими положениями. Обоснованность и достоверность прикладных результатов диссертации подтверждается результатами моделирования, апробации и промышленного внедрения предложенных методики и алгоритмов при проектировании устройств информационных комплексов.

**Апробация результатов исследования**. Результаты работы отражались в научных докладах, которые представлялись на следующих конференциях: на IX Всероссийской научно-практической конференции студентов, аспирантов, работников образования и промышленности на тему «Автоматизация процесса разработки встраиваемого программного обеспечения» (Омск, 2019); I Международной научно-практической конференции «Научный потенциал молодежи и технический прогресс» на тему «Подход к построению систем отладки и тестирования в современных вычислительных комплексах» (Санкт-Петербург, 2018); II Международной научно-технической конференции «Передовые технологии в аэрокосмической отрасли, машиностроении и автоматизации» (MIST-2019); Международном семинаре «Передовые технологии в материаловедении, машиностроении и автоматизации» (MIP-2019).

**Публикации по теме исследования**. По результатам исследований опубликовано 18 научных работ, в том числе 3 научные статьи в рецензируемых научных изданиях, рекомендованных ВАК при Минобрнауки России, 2 научные статьи в изданиях, индексируемых в международной реферативной базе данных Scopus, 2 патента на изобретения, 2 свидетельства о государственной регистрации программ для ЭВМ.

**Структура и объем диссертации**. Диссертация состоит из введения, четырех глав, заключения, списка использованных источников (119 наименований) и трех приложений. Общий объем работы 153 страницы, в том числе 134 страницы основного текста, включая 51 рисунок и 20 таблиц. Личный вклад диссертанта в работы, выполненные в соавторстве, состоит в разработке моделей, способов и алгоритмов имитации неисправностей и анализа реакции на имитируемые неисправности, а также представлении результатов исследований для опубликования.

Автор благодарит научного руководителя, д.т.н., доцента, профессора кафедры АСОИУ Денисову Л.А. за помощь при подготовке диссертационной работы. Автор выражает благодарность заведующему кафедрой АСОИУ, д.т.н., профессору Никонову А.В. за поддержку и к.т.н., начальнику отдела 5 АО ОНИИП Сидоренко К.А. за помощь в организации экспериментальных исследований.

# 1 ПРОБЛЕМА ПРОЕКТИРОВАНИЯ УСТОЙЧИВЫХ К ОТКАЗАМ И СБОЯМ УСТРОЙСТВ ИНФОРМАЦИОННЫХ КОМПЛЕКСОВ

В первой главе **рассматривается** комплекс проблем в области обеспечения устойчивости устройств информационных комплексов к отказам и сбоям, включающий проблему надежности ПО, проблему гармонизации отечественных стандартов в области проектирования цифровых устройств, проблему проявления системных ошибок при загрузке пользовательских программ в реальную аппаратную среду устройства.

## 1.1 Современное состояние проблемы обеспечения устойчивости к отказам и сбоям программно-аппаратных комплексов

Современные стандарты разработки и проектирования микропроцессорных устройств основаны на следующих методиках: FMEA (*Failure Mode and Effects Analysis*, для анализа видов и последствий отказов), FMECA (*Failure Mode, Effects, and Criticality Analysis*, для анализа эффектов режима отказа и критичности), FMEDA (*Failure Modes, Effects, and Diagnostic Analysis*, для диагностического анализа эффектов и режимов отказов). Стандарты находятся практически в перманентном системном кризисе, что отражается в их непрерывном изменении и создании новых стандартов [1, 7, 8, 9, 10, 11, 12, 13, 14, 88]. Это происходит в связи с постоянным ростом сложности программно-аппаратных комплексов и условий их применения, с одной стороны, и с проявлением свойства эмерджентности систем, которое не учитывается и не может быть учтено на этапе проектирования, с другой.

Актуальные стандарты, служащие руководством при создании цифровых устройств, опираются на процедуры определения видов, последствий и причин отказов с оценкой риска для проведения предупреждающих или корректирующих действий (FMEA) [79, 100], вычисления на основе FMEA-анализа показателей безопасности – числа приоритетности риска или критичности отказа (FMECA) [7, 55], нахождения по результатам FMECA-анализа частоты (интенсивности)

отказов для оценки надежности (FMEDA) [115]. FMEA- и FMECA-анализы прописаны в стандарте IEC 61508-7. Однако даже на уровне стандартов существуют противоречия в трактовке процедуры FMEA, который в большинстве стандартов рассматривается как качественный (не количественный) метод анализа (IEC 61508-7) [88], но в ГОСТе Р ИСО 26262-9-2014 [11] тот же FMEA уже становится количественным методом анализа вида и последствий отказов и может быть использован как количественный метод анализа для случайных отказов аппаратных средств в случае наличия сведений о количественных значениях интенсивности отказов (но в стандарте не поясняется, как именно). Так проявляется проблема гармонизации стандартов.

Программно-аппаратные устройства информационных комплексов согласно ГОСТу «Р МЭК 62061-2015. Безопасность оборудования. Функциональная безопасность» [13] должны обеспечивать устойчивость к отказам и сбоям. Устойчивость к отказам и сбоям определяется как способность подсистемы продолжать корректно выполнять требуемые функции при наличии отказов, сбоев или ошибок. При этом в ходе разработки нового изделия оно существует в единственном экземпляре в виде макетного образца или прототипа и на этом этапе жизненного цикла изделия является уникальным. Российский ГОСТ 27.002-2015 «Надежность в технике» [9] описывает расчёты показателей надежности только для технических объектов с известными характеристиками, поэтому применение стандарта для уникальных устройств ограничено в связи с большим временем сбора статистики и определением характеристик.

Процесс разработки безопасных программируемых систем описан в разделе С. 5.18 стандарта IEC 61508-7 (российский аналог ГОСТ Р МЭК 61508-7-2012) [12], который предполагает создание системы только для целей тестирования. Система имитирует поведение контролируемого оборудования, включая сочетание программного и аппаратного обеспечения со всеми входными данными для проверяемого устройства, которые возможны при реальной работе. Для тестирования аппаратного и программного обеспечения используется методика «инъекции неисправности» (*fault injection*) [76, 80, 81, 85, 96], с помощью которой

можно смоделировать любой вектор последствий неисправности и любое количество этих векторов, а также сделать программное обеспечение для многократного воспроизведения сценария векторов последствий неисправности или имитации неисправности, что предусмотрено в международных стандартах [119] IEC 61508, EN 50128 и ISO 26262 и их отечественных аналогах. Перечисленные стандарты не определяют объёмы испытаний, требования к аппаратуре и технологии проведения испытаний. Технология для проведения испытаний по имитации неисправностей может быть создана только на основе аппаратуры имитации. Для испытания на устойчивость к отказам и сбоям происходит создание системы только для целей тестирования, которая имитирует поведение оборудования под контролем.

Альгирдас Авиженис, основатель и первый председатель технического комитета по отказоустойчивым вычислениям, консультант и участник ряда разработок для ВВС, ВКФ, НАСА и лауреат престижной премии в области разработки компьютерных архитектур Эккерта-Мочли, подтверждает [82, 83, 84, 109] необходимость исследования системы методом имитации неисправностей, поскольку экспериментальное прогнозирование надёжности путем исследования опытного образца, а не имитационной модели требует больших усилий на его создание, но позволяет избегать неточностей, которые могут возникнуть при априорном постулировании эффектов неисправностей в имитационной модели системы.

Исследованию проблемы робастности (являющейся следствием проявления нежелательных эмерджентных свойств при проектировании программно-аппаратных систем) посвящен ряд работ [28, 42, 116]. Для ее проявлений в процессе проектирования устройств применяют специальные инструменты, в том числе имитирующих неисправности [3, 20, 54]. Одним из направлений исследований с помощью искажения входных данных (техника фаззинга) имеет актуальность практически для всех современных микропроцессорных устройств с программным обеспечением.

За последние тридцать лет сложность программного обеспечения информационных комплексов выросла на несколько порядков [23]. Тем не менее, отечественная промышленность использует устаревшие технологии для проектирования цифровых систем различной степени сложности [5]. ПО не выделяется в отдельный «смысловой» элемент системы. Если подходить к программным продуктам, выпускаемым современными производителями, ни одна коммерческая компания не несет ответственности за эксплуатацию уязвимостей и недокументированных возможностей системы.

Современные микропроцессоры с встроенным программным обеспечением, разработанные в зарубежных компаниях, часто содержат программно-аппаратные «закладки», предназначенные для отладки и тестирования системы, но при целенаправленных действиях могут использоваться для уничтожения либо приостановки работы информационных систем.

В методологии разработки отказоустойчивого ПО исследователи выделяют три основные группы проблем [29, 30]:

– отсутствие единой методологии создания отказоустойчивого ПО;

– отсутствие единой методологии тестирования отказоустойчивого ПО;

– отсутствие единого подхода к анализу проблемной области.

Альтернативный взгляд на эту проблему предполагает, что такой проблемы не существует в принципе. Логика данного подхода заключается в том, что:

– программы и аппаратура представляют единый комплекс, в котором аппаратными средствами возможна компенсация отдельных отказов ПО, и исследовать надёжность отказоустойчивого ПО, игнорируя это обстоятельство, бессмысленно;

– существует большой класс аппаратурных средств контроля за ошибками отказоустойчивого ПО, которые во многом компенсируют отказы ПО;

– отказоустойчивое ПО может содержать только ошибки, не выявленные на этапе отладки, а отказ системы — следствие комплексного влияния отказов, сбоев и ошибок, воздействующих на систему, оценить которое теоретически не представляется возможным в связи с неисчерпаемым количеством вариантов;

– не решена проблема робастности ПО — устойчивости к различным наборам входных данных, которая делает бессмысленной создание единой методологии создания отказоустойчивого ПО.

Выход из этой группы проблем заключается в том, что испытания проводятся для всей аппаратуры комплексно с помощью средств имитации неисправностей, выделяя ПО как отдельный компонент системы. Моделирование процессов устройств информационных комплексов является одной из основных задач, решаемых на начальном этапе проектирования системы. При разработке стратегии управления информационными системами возникают значительные трудности, связанные с тем, что имеющиеся в этой отрасли международные стандарты не содержат алгоритма реализации предписанных требований по обеспечению надежности и качества ПО для проектируемых систем [34, 54].

Актуальной проблемой является гармонизации государственных стандартов с международными стандартами, создание единой терминологии для описания проблемной области, что позволит избегать неточностей и дублирования при создании новых цифровых устройств [52, 62, 65]. Поэтому разработка подходов к имитации неисправностей при проектировании устройств информационного комплекса будет использовать набор технологий, который предоставляет допустимое решение в предметной области с помощью применения системного анализа. Таким образом, для обеспечения работоспособности прототипов микропроцессорных устройств требует применения системного анализа.

## 1.2 Анализ методов повышения отказоустойчивости микропроцессорных устройств

Современные микропроцессорные устройства используют чрезвычайно устойчивые технологии проектирования. Промышленные архитектуры проходят тестирование на заводах-изготовителях [2, 4]. Сегодня в мире основными элементами цифровой техники являются интегральные полевые микросхемы, такие как программируемые массивы логических элементов, микропроцессоры,

различные элементы памяти и т.п. Основные производители больших и сверхбольших микросхем представляют фирмы Altera, Xilinx, Atmel, STM32 и др. Все изготовители обязаны подтверждать качество своей продукции, в частности показатели надёжности [54, 68, 72, 74].

Тем не менее, для каждого микропроцессорного модуля, вышедшего с конвейера, возможно использование совместно с другими микросхемами и программным обеспечением. Современные микропроцессоры поддерживают специальные алгоритмы по восстановлению работоспособности после возникновения отказов и сбоев. Проектируемое устройство на базе микропроцессора относится к сложным вычислительным системам, поскольку состоит из отдельных элементов, взаимодействующих между собой и с окружающей средой (прием и передача информации), а также выполняющих специальные функции. В системе при загрузке программ в целевую аппаратную среду проявляется свойство эмерджентности [77, 80], что фиксируется с помощью неожиданных отказов и сбоев. Для диагностики отказов микропроцессорные архитектуры содержат специальный блок обработки отказов, который различает следующие типы ситуаций:

1. Тяжелые отказы (*Hard Fault*).
2. Отказы системы управления памятью (*MemManage Fault*).
3. Отказы программы (*Usage Fault*).
4. Отказы шины (*Bus Fault*).

Микропроцессорное ядро содержит сохранение контекста вызовов инструкций, т.е. при возникновении отказа регистры процессора сохраняются, после чего производится детальный анализ инструкций и вызываемых адресов программы, чтобы установить причину возникновения. Тем не менее, существуют ситуации, в которых восстановления контекста выполнения микропроцессора невозможно, поэтому используются специальные средства диагностики, включающие аппаратные блоки контроля и верификации сигналов системных интерфейсов. Отличие аппаратных отказов и сбоев от программных является для

микропроцессорных устройств важной задачей, поскольку проявление последствий программного отказа возможно в виде аппаратной неисправности.

В связи с тем, что аппаратные компоненты программно-технических систем относятся к классу высоконадёжных изделий (соответствие стандарту ISO 9000), выявление дефектов программного обеспечения в аппаратной среде является наиболее актуальным. Особенно с учетом того, что известные методики испытаний программного обеспечения для обнаружения неисправностей в сложных распределенных информационных комплексах недостаточно эффективны и требуют усовершенствования [29, 61].

Устойчивость к отказам и сбоям для микропроцессорного устройства рассматривается совместно с интегрированными микросхемами и программным обеспечением, необходимым для его функционирования в информационном комплексе [71] и должна соответствовать уровню требований для информационного комплекса, то для них предъявляются требования к живучести:

$$X^* = \min\{X : \Phi_{\overline{X}} = 0\}, \tag{1.1}$$

где $X^*$ – минимальное подмножество, $\overline{X}$ – подмножество разрушенных элементов, $X$ – дополнительное подмножество, $\Phi_{\overline{X}}$ – условная вероятность отказа. Свойство безопасности для работы микропроцессорного устройства обеспечивается за счет ряда ограничений, которые направлены в рамках теории рисков.

$$\min_{\psi}\{C(\psi) : R(\psi) \geq R_{required}, C(\psi) \geq C_{required}\}, \tag{1.2}$$

где $\Psi$ есть конфигурация системы, $C$ – ее стоимость, $R$ – показатель надежности, а $S$ – показатель безопасности.

Практически все современные технические устройства являются цифровыми. Развитие стандартов для цифровой техники использует технологическую базу имитации неисправностей для функциональной безопасности и анализа критичности выполнения режимов [31, 103]. Представленный ряд стандартов применим и к микропроцессорным системам, которые также являются частью более сложных комплексов [119].

Синтез стандартов для цифровых систем, примененный к микропроцессорным устройствам, позволяет выделить необходимые требования к проектированию (таблица 1.1). Среди них работоспособность и устойчивость к отказам и сбоям. Сформированные в рамках стандарта ISO 26262 предложения по разработке включают практически все подходы рассмотренных стандартов.

Таблица 1.1 – Методики и стандарты в области повышения отказоустойчивости

| ***Стандарт / методика*** | ***Тип/ назначение*** | ***Год введения*** | ***Область применения*** |
|---|---|---|---|
| *MIL-STD-1629, МЭК 271-85* | FMEA | 1949, 1985 | Процедуры выполнения анализа отказа режимов, эффектов и критичности для устройств |
| *EN 50128, IEC 62279* | FMEA/ Fault Injection | 2015 | Железнодорожные приложения. Системы связи, сигнализации и обработки данных |
| *ISO 9001* | FMEA | 2015 | Системы менеджмента качества |
| *IEC 61508* | FMEA | 2012 | Функциональная безопасность систем электрических, электронных, программируемых, связанных с безопасностью |
| *ГОСТ 27.310-95* | FMEA | 1995 | Анализ видов, последствий и критичности отказов |
| *ISO 26262* | FMEA/ Fault Injection | 2011 | Функциональная безопасность дорожных транспортных средств |
| *Chaos Engineering* | NETFLIX | 2011 | Экспериментальная методика для смягчения последствий сбоев |
| *FMEA* – методология проведения анализа видов и последствий отказов; *Fault Injection* – метод тестирования, который помогает понять, как реальная система ведет себя при необычных нагрузках. | | | |

Поскольку стандарт ISO 26262 рассматривает функциональное оборудование для безопасности дорожных транспортных средств, которые включают микропроцессорные системы, то данный стандарт применим для испытания микропроцессорных систем информационных комплексов. Особенно при условии

наличия в информационном комплексе транспортных средств обмена информацией, к которым требования стандарта обязательны.

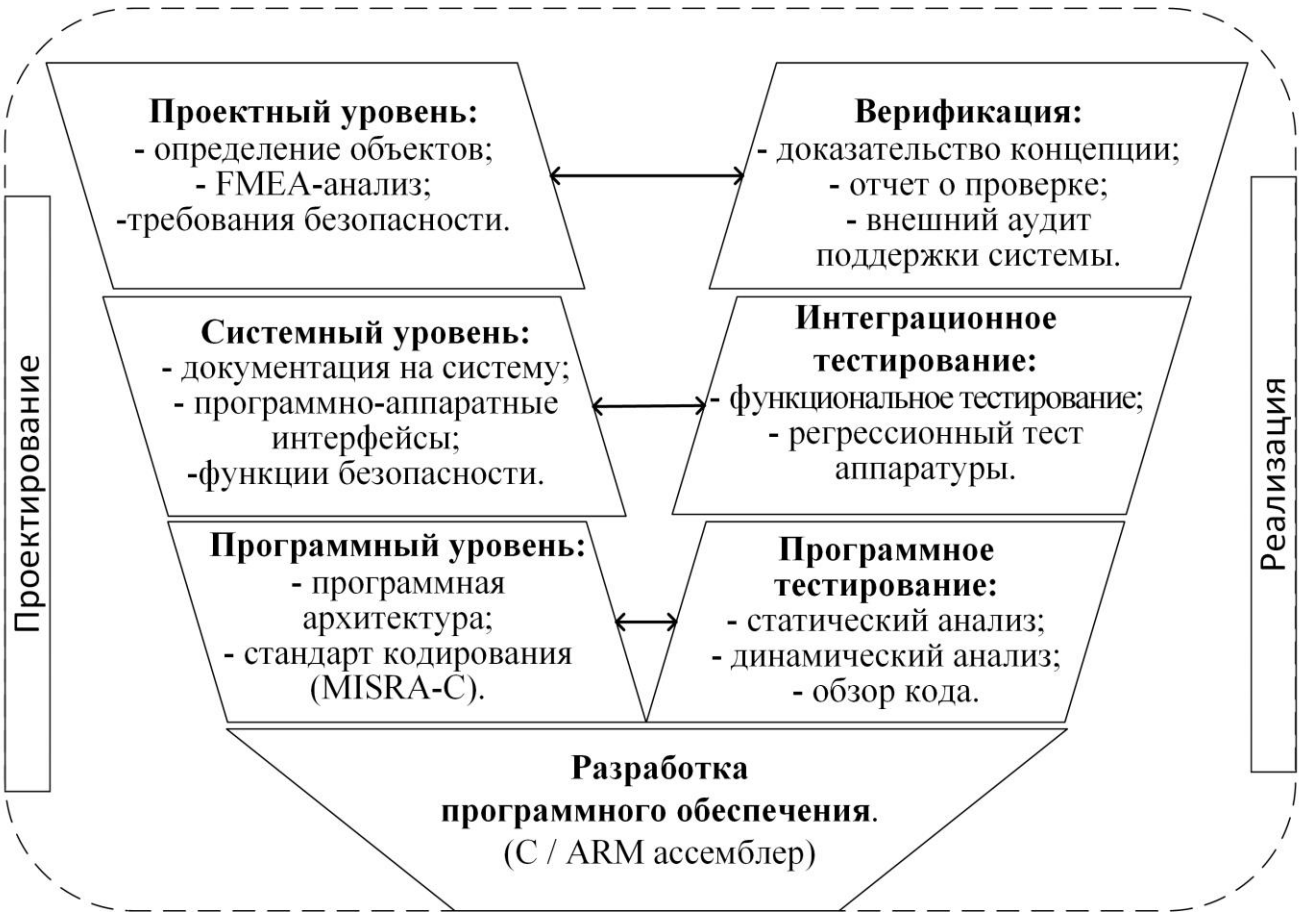



Рисунок 1.1 – Соответствие уровней и процедур проектирования циклу функциональной безопасности, регламентируемому стандартом ISO 26262

В международной практике методика FMEA трактуется как качественный метод анализа. Но в стандарте ISO 26262-9 (п. 8.2) FMEA может быть использован как количественный метод анализа для случайных отказов аппаратных средств при наличии дополнительных сведений о количественных значениях интенсивности отказов. При этом в стандарте не поясняется, как именно. FMEA является базовым анализом для проведения FMECA- и FMEDA-анализов. FMECA и FMEDA являются количественными анализами для оценки безопасности и надежности соответственно. FMEDA — это достаточно новый стандарт, и в старую редакцию IEC 61508-7 он не включен. Имитация неисправностей (*fault injection – FI*) также присутствует в IEC 61508-7 (пп. B6.10 и C5.6) как самостоятельный качественный (не количественный) анализ. FMEDA используется для оценки системных отказов

(по которым есть статистическая наработка), а FI — для анализа случайных (спорадических) отказов. FMEDA используется в первую очередь для покупных изделий.

Электронные компоненты, к которым можно применить эти методы, рассматриваются как частный случай использования вероятностных моделей типа «черный ящик». Для расширения технологии (FI) разрабатывается специализированная аппаратура, например, облучающая определенные участки печатной платы сильной радиацией. Подключаемые имитаторы используют интерфейсы системы либо выводы элементов на монтажных платах. Встроенные имитаторы являются элементом самой системы и не подлежат исключению, т.е. являются неудаляемой частью исследуемой системы и могут использоваться по другому назначению, кроме FI.

К встроенным относятся и программные методы Software Implemented Fault Injection (SWIFI) [90]. Имитация ошибок может осуществляться чисто программным способом либо с помощью аппаратного обеспечения, которое в свою очередь делится на стандартное и специальное. К стандартному относятся внутрисхемные эмуляторы и средства для отладки (JTAG-интерфейс) [97].

Таким образом, создание имитаторов неисправностей, специализированных для микропроцессорных устройств информационных комплексов, является актуальной задачей, поскольку имитация отказов и сбоев позволяет исследовать проектируемую систему и выявить ошибки программ и аппаратные дефекты на этапе макетирования до выхода в серийную разработку, что повысит качество конечного изделия.

Имитаторы неисправностей для микропроцессорных систем разделены на два больших класса: имитаторы непосредственно самих неисправностей и их последствий. По способу внесения разделяют программные (*software injection*) и аппаратные (*hardware injection*) средства. По расположению по отношению к исследуемой системе выделяют внешние, встроенные и подключаемые. Внешние используют режимы облучения. Имитатор для микропроцессорной системы должен обеспечивать следующий набор функций [85, 92, 96]:

– обеспечение в период испытаний всех входных сигналов системы, которые будут существовать, когда система проходит проверку;

– обеспечение выходам системы пути, которые адекватно представляют контролируемое оборудование;

– обеспечение возможности управления входами для обеспечения любых возмущений, с которыми должна справляться тестируемая система.

Стандарт IEC 61508 [103] определяет у имитаторов неисправностей такие функции, как проверка и верификация. Построение имитаторов для микропроцессорных устройств на базе серийно выпускаемых средств вычислительной техники является актуальной задачей для новых поколений средств автоматизации, в рамках которых решаются задачи:

1. Проводится анализ прототипов автоматизированной системы испытаний (АСИ) и разрабатываются обобщенные модели таких систем, основанные на имитации неисправностей и имитации ошибок.

2. Исследуется систематизация способов имитации ошибок и механизмов их реализации, на основе которых производится целенаправленный выбор этих способов при синтезе новых средств имитации неисправностей.

3. Разрабатывается модель программируемых средств имитации неисправностей для испытаний, позволяющая автоматизировать настройку на тип имитируемой неисправности и тип неисправного компонента, а также обладающая свойствами безопасности и сопрягаемости.

4. Разрабатываются модели ошибок, позволяющие обоснованно выбрать тип искажений сигналов в системе.

5. Производится выбор способов имитации с учетом разработанной модели ошибок и новые механизмы их реализации, учитывающие специфику объекта исследования.

6. Разрабатывается язык для описания процессов имитации последствий неисправностей, позволяющий формализовать процесс проектирования имитаторов неисправностей.

7. Осуществляется выбор обобщенной архитектуры функционально-ориентированного процессора для имитации последствий неисправностей, на основе которой предлагается инженерная методика для синтеза имитаторов неисправностей.

## 1.3 Анализ средств обнаружения отказов и сбоев устройств информационного комплекса

Для обнаружения отказов и сбоев устройств информационного комплекса применяются как встроенные средства составных устройств, так и анализ статистики комплекса. Это позволяет выявить выход из строя устройств комплекса. Стоит отметить, что набор мер для обеспечения устойчивости к отказам и сбоям будет значительно отличаться в зависимости от сферы применения комплекса. Для космических и атомных систем применяются наиболее строгие методики проверки на отказы и сбои, чем для информационных комплексов [71, 112].

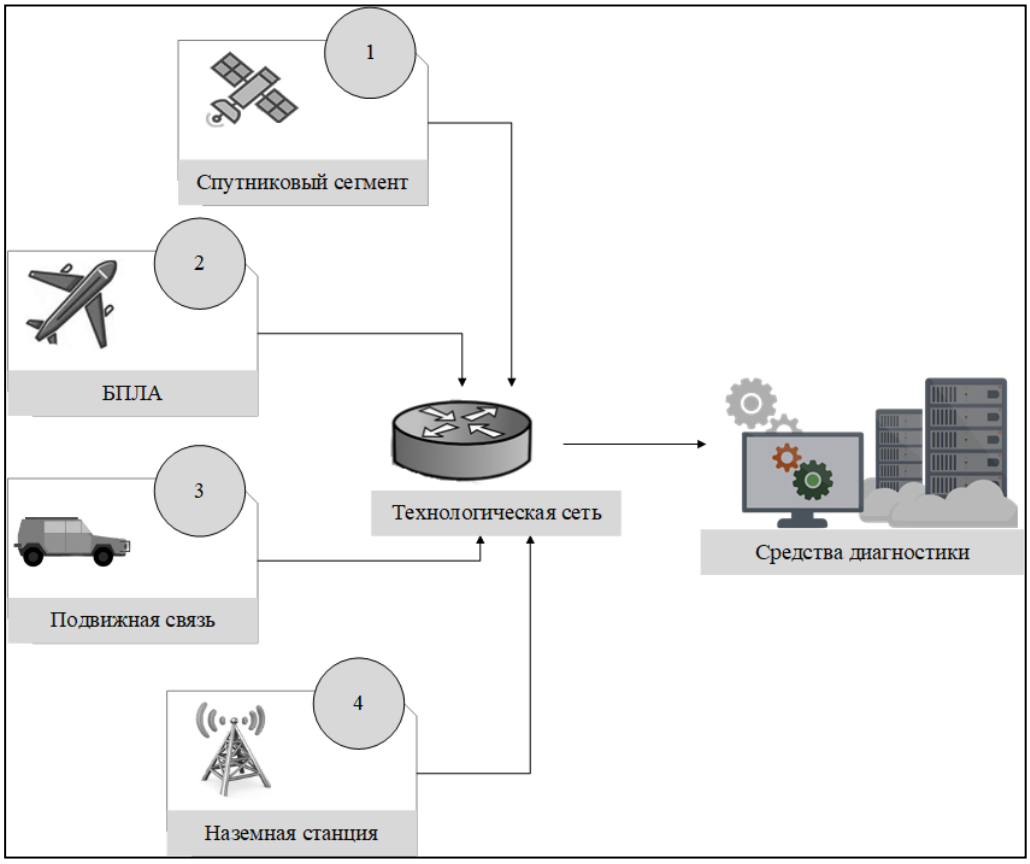



Рисунок 1.2 – Топология программно-аппаратных компонентов сети

Чтобы обеспечить проверку на отказы и сбои применительно к устройствам, содержащим ряд микропроцессорных устройств, требуется определить информационный портрет контролируемой системы. Проведение анализа сложных или многофункциональных систем осуществляется с помощью нескольких методов анализа. На практике использование комбинаций нисходящего и восходящего анализов является весьма эффективным и позволяет обеспечить комплексное рассмотрение для определения показателей анализа существующих систем. Информационный портрет представляет собой результат анализа поведения систем в условиях имитируемых отказов и сбоев, где проводится сбор и обработка статистических данных, на основе которых предлагаются оценки параметров комплекса [21, 53, 54].

Для оценки параметров в комплексе часто используется статистический анализ с помощью методов математической статистики и машинного обучения.

Рисунок 1.3 –Визуализация зависимости признаков информационного комплекса

Статистический анализ условий возникновения событий используется для установления корреляции между выходом устройств из строя в процессе работы [24, 33]. В результате моделирования устанавливается, какие показатели наиболее информативны для установления неработоспособности конкретного узла.

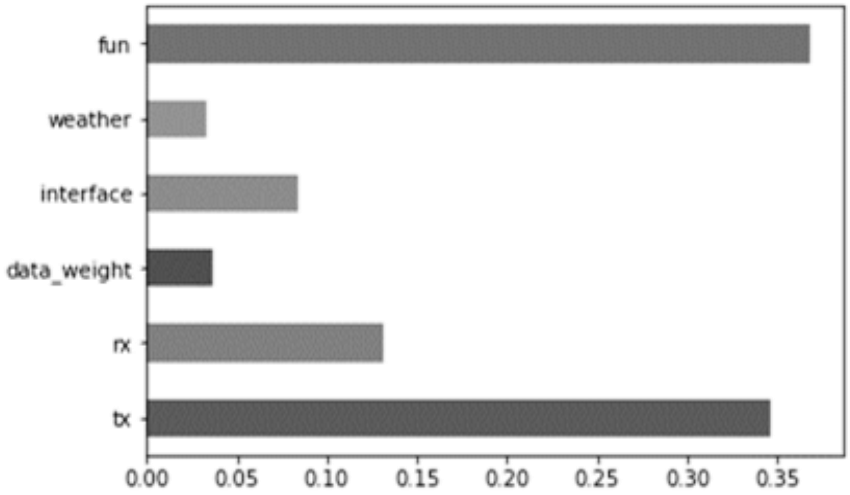



Рисунок 1.4 – Градация зависимости результата работы от статистических признаков

Для подтверждения аппаратных отказов в информационном комплексе требуется использование модулей аппаратного контроля, определяющих технические неисправности компонент. Одним из эффективных подходов является сравнение сигналов на входах и выходах и параметрическая регуляция модуля. При отклонении модуля от установленного порядка фиксируется неисправность, после чего информация передается рабочей станции. Применение таких модулей актуально для основных подсистем микроконтроллера: системы управления внутренней памятью, шин данных, ввода-вывода.

Для выявления аппаратных отказов требуется использование модулей контроля, определяющих технические неисправности компонент. Отказы и сбои определяются путем сравнения сигналов на входах и выходах и параметрической регуляции модуля [35]. При отклонении модуля от установленного порядка фиксируется неисправность, после чего информация передается рабочей станции. Применение таких модулей актуально для основных подсистем микроконтроллера: системы управления внутренней памятью, шин данных, ввода-вывода, аппаратного таймера.

Использование имитации неисправностей для целей тестирования информационного комплекса является сложной задачей и предполагает создание комплекса автоматизированных средств, а также специального программного обеспечения. Теоретически наибольшей точностью обладают экспериментальные

методы, которые являются основным способом подтверждения требуемых по техническому заданию показателей работоспособности с заданной точностью и достоверностью. Экспериментальная оценка показателей может быть реализована только после создания исследуемой системы и является последним и самым важным этапом в процессе разработки информационных комплексов. Экспериментальные методы оценки надежности, связанные с организацией специальных испытаний, можно разделить на четыре группы [42].

1) Методы, использующие естественные факторы старения при исследовании систем, главный среди которых - время. При нормальных условиях эксплуатации могут быть исследованы сравнительно недорогие системы временем наработки на отказ менее одной тысячи часов. Для уникальных систем с временем наработки более десяти тысяч часов метода первой группы совершенно не подходят.

2) Методы, использующие искусственные факторы, вызывающие старение элементов, температура, напряжение питания, влажность, давление, вибрация и т.п. Основная сложность при испытаниях устройства состоит в том, что практически невозможно подобрать параметры, при которых все разнородные компоненты системы ускоренно старели с близкими интенсивностями. Вследствие этого результаты ускоренных испытаний сложного оборудования, каким является микропроцессорное устройство, всегда будут отличаться от результатов естественного старения.

3) Методы, использующие искусственное введение неисправностей компонентов устройства, основанные на разрушении структуры компонентов системы или связей с помощью введения отказов и/или искажении сигналов в системе с помощью генераторов помех - введение сбоев. Трудности, связанные с введением различных типов отказов и сбоев в реальном масштабе времени для большого числа компонентов, а также выход из строя оборудования системы делают невыгодным применение данной группы методов.

4) Методы, использующие имитацию неисправностей (последствий неисправностей) компонентов устройства, основанные на имитации отказов и сбоев путем искажения сигналов в системе с помощью специализированных

устройств (имитаторов неисправностей), позволяют точно определить причину отказа системы, поскольку время и место имитируемой неисправности известно, а состояние устройства в момент имитации неисправностей может быть определено или задано. Таким образом появляется возможность не только оценить достигнутые показатели устойчивости устройства к отказам и сбоям, но и проводить направленную доработку уникальных, дорогостоящих систем с тем, чтобы повысить их уровень надежности.

Процесс имитации неисправности заключается в том, чтобы внести неисправности в реальной аппаратуре, оценить реакцию системы на внесённые искажения и собрать статистическую информацию о том, сколько раз система откажет [72, 73]

$$p = \frac{n(t)}{N}, \tag{1.3}$$

где $p$ – вероятность безотказной работы, $n(t)$ – число отказов за время $t$, $N$ – общее число отказов.

Отказы элементов системы могли не приводить к её отказу, если система переходила в работоспособное состояние за приемлемое время или приводить, если после восстановление работоспособности достигалось только за счет диагностики и ремонта. Имитация неисправностей в реальной системе принципиально модифицирует средства исследования системы и улучшает её качество на этапе сдачи в эксплуатацию.

## 1.4 Тенденции в области разработки надежного программного обеспечения

Современное программное обеспечение решает все больше задач, которые ранее выполнялись с применением аппаратных средств. Программные компоненты используются практически во всех отдельных микросхемах, которые включены в микропроцессорные модули и подключены к ним. На замену устройствам с разделением программ на разные аппаратные модули приходят микропроцессоры, позволяющие выполнить единую программную реализацию алгоритмов и интерфейсов, которые должны соответствовать повышенным требованиям

устойчивости к отказам и сбоям. При увеличении объема пользовательских программ произошел рост программных ошибок, что привело к изменению и усложнению методов и средств отладки, диагностики и верификации [64, 65].

В 80-90-е годы были попытки применить вероятностные модели для оценивания надежности программного обеспечения, ошибок оператора, а затем и показателей безопасности. Модели надежности ПО используются для анализа и поиска дефектов программ в процессе разработки и эксплуатации. Практика разработки ПО предполагает приоритет задачи обеспечения надежности над задачей ее оценки [5]. Ситуация выглядит парадоксально: совершенно очевидно, что прежде чем обеспечивать надежность, следует научиться ее измерять. Но для этого нужно иметь практически приемлемую единицу измерения надежности ПО и модель ее расчета. Экспериментально модели оценки надёжности ПО отличаются от вычисленных значений.

Тем не менее надежность программ рассматривается [5, 51] как компонент качества программного обеспечения. Применение моделей расчёта часто не обосновано для простых программ. Основное применение для расчётов на сегодняшний день получили модели, которые используют структуру времени и эмпирические модели. Модели структуры времени учитывают поэтапное тестирование ПО с применением оценки за определенный временной промежуток. Программы для микропроцессорных устройств относятся к сложному программному обеспечению (критерий сложности – число строк программы) [10]. Подтверждением практический значимости эмпирических моделей является появление специальных средств симуляции, направленных на практическую эксплуатацию программ для цифровых электронных систем. Применение данных моделей используется современными САПР для расчёта надёжности ПО с использованием логико-вероятностных методов построения графических моделей безопасности, деревья событий и деревья отказов. Использование аппарата математической логики в вычисления производится для крупносерийных контроллеров с малым объемом ПО, к которым применимы формальные модели

надежности. В основном данные комплексы предназначаются для крупных объектов АСУ и промышленных комплексов [65].

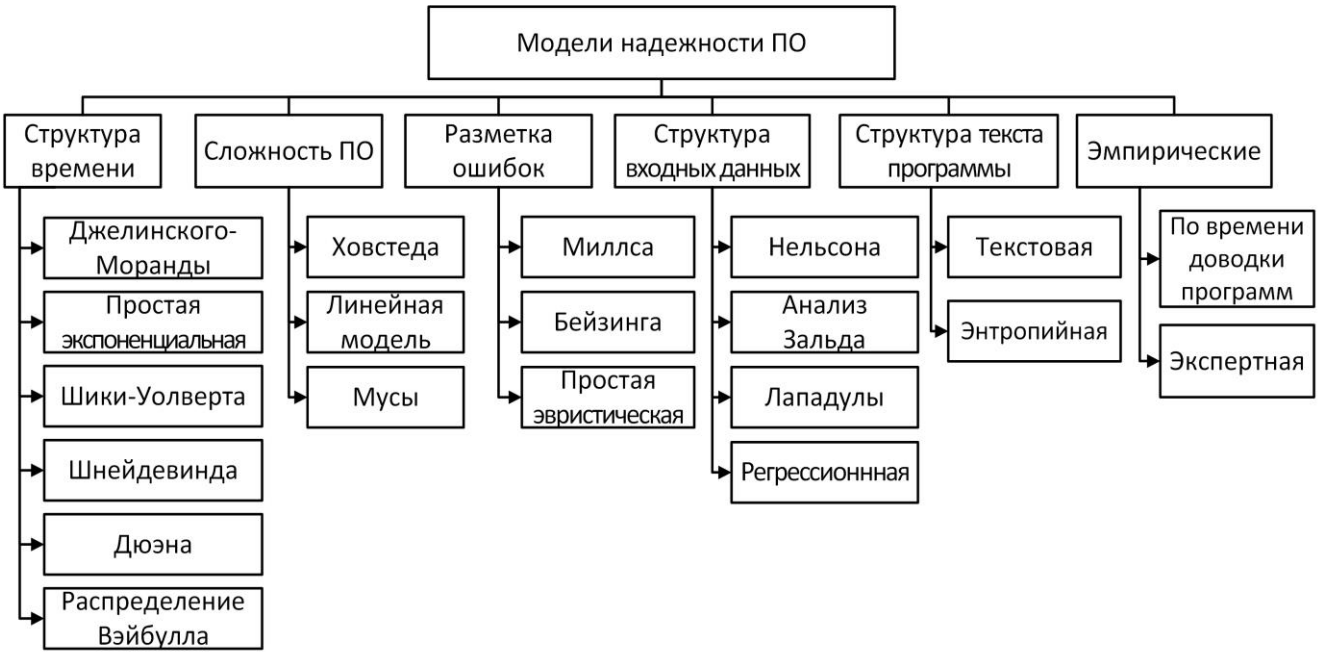



Рисунок 1.5 – Модели надежности ПО

Тестирование программ превратилось в самостоятельную область исследования, которая содержит множество методов и алгоритмов, имеющих различную область применения и степень интеграции [66]. Большинство из них хорошо применимы при моделировании, но практическое применение моделей остается актуальной задачей. Это подтверждает многообразие видов тестирования программ: модульное, интеграционное, мутационное, регрессионное, сквозное, системное. Тестирование использует аппаратные средства для контроля, диагностики и имитации ошибок. Так, процессоры *Intel* используют технологию «*Intel Packet Trace*» [107] для отладки программ совместно с аппаратурой, микропроцессорные системы стандартно предоставляют доступ к регистрам, памяти и стеку для анализа состояния программы на базе промышленного интерфейса отладки JTAG [97, 99].

Одной из тенденций развития программ можно отметить уделение дизайну программ первоочередного места и появления специальной профессии – архитектора программного обеспечения, целью которого является минимизация ошибок программ за счет управления сложностью взаимодействия модулей в

программе и качественному покрытию программы тестами. Современные программы также используют стандарты в области архитектуры программного обеспечения, что позволяет комбинировать структурные элементы программ в блоки, которые легче тестировать за счет применения стилей, которые представляют так называемые «best practices» [37]. В современной разработке программ используется ряд шаблонов архитектур программного обеспечения. Для микропроцессорных устройств часть архитектур является избыточными или нецелевыми, поскольку ориентированы на веб-технологии или серверное оборудование. Наиболее актуально применение концепции «чистой архитектуры», разработанной Робертом Мартином [39]. Для спроектированной архитектуры часто используют определение коэффициента надежности ПО [29]

$$R = \sum_{j=1}^{M} \sum_{i=1}^{N_j} PU_{ij} R_{ij} \,, \tag{1.4}$$

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.

Разработка программ с увеличением объема кодовой базы претерпевает изменения в плане упрощения используемых языков программирования. Это происходит с помощью конструирования новых предметно-специфицированных языков (*DSL*) [19, 20] и упрощения программного кода для разработчика. Тем не менее, переход к новым программным языкам содержит ряд недостатков, которые выявляются разработчиками по всему миру. Исправление ошибок языков является сложной задачей, особенно с учетом наличия огромной кодовой базы и работающих программ. К тому же такие решения обладают проблемами, связанными с точностью трансляции разработанных программ в ассемблер архитектуры микропроцессорных модулей. Также можно отметить ошибки в компиляторах и интерпретаторах предметных языков [23], которые искажают семантику программы.

Современные исследователи рассматривают программное обеспечение как динамическую систему, свойства которой до конца не определены. Поэтому методики и алгоритмы эффективного исследования программ и получение практических оценок по эксплуатации на реальных устройствах важно для разработки качественного ПО. Понятие качества ПО является многоаспектным, исследователи не пришли к единому терминологическому определению. Одно из направлений упрощения и систематизации разработки программного обеспечения в части уменьшения числа программных ошибок производится с помощью внедрения стандартов кодирования.

Существующие стандарты *JPL Institutional Coding Standard*, *MISRA, HICPP*, *Google C++ Style Guide* позволяют уменьшить число ошибок за счет ограничения в использовании инструкций для языка программирования [26], что позволяет допускать меньшее число ошибок на стадии кодирования, а не тестирования. В число таких практик входит применение при разработке программного обеспечения контроля уровня объектно-ориентированных абстракций либо уровня абстракций предметной области. Появление технологий контроля версий (*Git* – система совместной разработки), автоматизированной инфраструктуры виртуализации (*Docker* – виртуализация программных компонент) совместно с методиками тестирования, обзора кода, рефакторинга и статического анализа позволяют решить большую часть проблем при написании кода [29, 39]. Тем не менее развитие средств отладки и диагностики для устройств информационных комплексов происходит гораздо медленнее в связи с коммерческими и специальными ограничениями, а также использованием собственных микропроцессорных решений для применения в разных эксплуатационных условиях, где существуют ограничения по энергопотреблению и используемой памяти [45].

В связи с чем используются языки 70-80-х годов, такие как язык С, который обладает рядом неоспоримых преимуществ, но не содержит модулей контроля и диагностики, как в языках Python, C# и *Java*, вследствие чего разработчики за период своей деятельности занимаются написанием и отладкой стандартных в

других языках модулей или интеграцией сторонних библиотек, которые могут содержать ошибки и требуют дополнительного времени на отладку. Также заимствованные решения часто работают не оптимальным образом, что заставляет программиста пробовать новые и новые программные библиотеки, соблюдая баланс между пониманием их внутреннего устройства и скоростью разработки [63].

Тенденцией в области разработки программного обеспечения является автоматизация процессов проектирования на всех этапах жизненного цикла программ. Автоматизация затрагивает процессы верификации, тестирования и контроля качества программного кода. Современные сервисы совместной разработки поддерживают процессы непрерывной интеграции и непрерывной поставки программного продукта (CI/CD) [87]. Проектирование конструкции и выпуск новых устройств длится большую часть жизненного цикла проекта. В связи с этим требуется эмулировать взаимодействие с неизготовленными аппаратными модулями. Таким образом, программист работает с двумя программами – для реального устройства и для программного эмулятора.

Реальное устройство может выходить из строя, поэтому необходимо поддерживать версии программного обеспечения как для эмулятора, так и для устройства. Процесс разработки состоит из следующих этапов жизненного цикла:

1. Разработка функционала модуля;
2. Объединение модулей в единой архитектуре;
3. Тестирование функциональности компонента и системы в целом;
4. Корректировка программы в результате аппаратной корректировки.

Для совместного использования версий продукта при разработке встраиваемого ПО используется система контроля версий. Главной проблемой разрабатываемых программ является нарушение алгоритмов работы при аппаратных корректировках и отказах. Чем раньше обнаружена проблема, тем легче производить корректировку версии ПО. Для проверки версии ПО и выявления несоответствий выполняется сборка, тестирование и корректировка программ. Для встраиваемых систем эти процедуры не автоматизированы [95]. Для

ускорения процесса разработки предлагается использование автоматизации проектирования при выполнении следующего набора процедур:

– автоматическая сборка проекта;

– автоматическая проверка проекта;

– автоматическое программирование устройства;

– автоматическое тестирование на базе тестовой инфраструктуры;

– предоставление результатов тестов на отказы и сбои системы.

Автоматизация сборки проекта возможна при использовании сервера непрерывной интеграции (на базе средств *Docker*). Для управления предлагается применение систем управления репозиториями (к примеру, *Gitlab*). На рисунке 1.6 приведен цикл обновления версий программ с помощью данного инструмента.

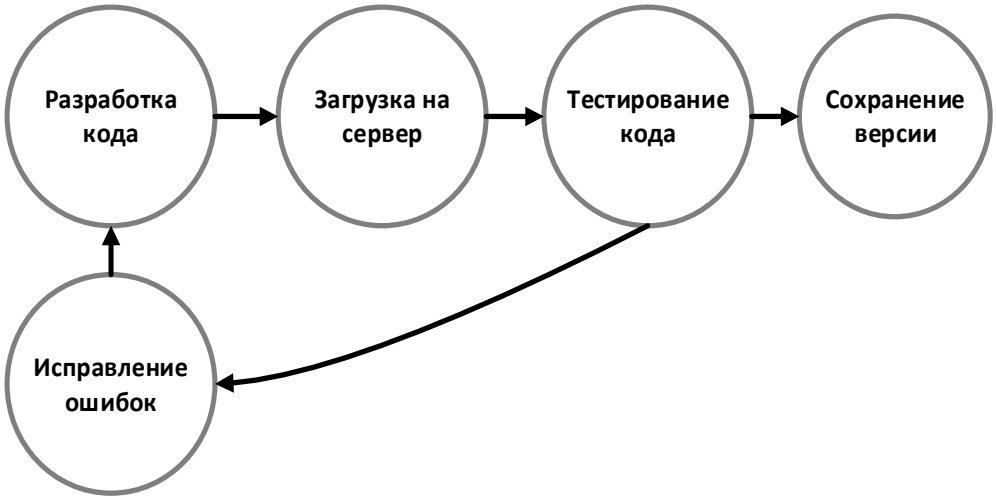



Рисунок 1.6 – Основные стадии автоматического обновления версий программы

Для разработки ПО осуществляется автоматическое тестирование на сервере и контроль программы без участия разработчика. При этом на поиск ошибок и коррекцию изменений уходит меньше времени по сравнению с ручным поиском, даже с учетом времени проектирования тестов.

Разработка ПО при использовании данной среды состоит из следующего набора этапов:

– подготавливается программный образ для разработки проекта на сервере. Образ загружается разработчиком на ПК, поэтому у всех разработчиков однообразная среда проектирования, доступная удаленно;

– внутри проекта существует два программных продукта, которые находятся в параллельных версиях проекта (для эмулятора и для реального устройства);

– при разработке программы алгоритмы проверяются с помощью модульных, интеграционных и регрессионных тестов. При загрузке на сервер программа проходит набор тестов для реальной среды и эмулятора, при наличии ошибок возвращается на исправление автору последних изменений.


Рисунок 1.7 – Система автоматизированного тестирования программ

Часть интеграционных и нагрузочных тестов требует наличия у разработчика специальной инфраструктуры, позволяющей выполнить замену объектов, взаимодействие с которыми необходимо для проверки сложных алгоритмов устройства. В инфраструктуру входят специальное оборудование, работающее в автоматическом режиме. Задача таких устройств – генерация сигналов и эмуляция протоколов для обмена с тестируемым устройством. Поскольку большинство устройств обладает интерфейсами подключения к ПК по *USB* или *Ethernet,* существует возможность программного запуска целой системы для имитации полноценного взаимодействия с реальным устройством. Система тестирования

организует выполнение действий по запуску в автоматическом режиме. Запуск производится с помощью этапов:

– выбрать последнюю версию программы с пройденными модульными тестами, автоматически собрать программы и с помощью отладочного оборудования, работающего по интерфейсу диагностики, загрузить версию программы в процессор встраиваемой системы;

– запустить подключенное оборудование с сигнальными и кодовыми последовательностями для интеграционного и регрессионного тестирования; сохранить отчет о тестировании для анализа разработчиками.

Разработка программ для современных устройств предполагает цикл работы, которой производится на протяжении всей разработки, опытной эксплуатации и поддержки. Цикл разработки включает непосредственно разработку архитектуры, кодирование ПО, отладку, тестирование, приемку, доработку ПО. Данные этапы разработки варьируются в зависимости от вида программного продукта. Разработка ПО для микропроцессорных устройств обладает дополнительными стадиями, которые обусловлены наличием аппаратных средств, которые тестируются совместно с программой. Для повышения качества разрабатываемого программного обеспечения используются гибкие методологии проектирования, которые нацелены на минимизацию времени существования невыявленных ошибок программ.

На практике исследования программ хорошо зарекомендовал подход *fault injection* [77], т.е. имитации отказов, который также реализуем с помощью программного обеспечения. Данный подход использует представление о том, что все критически важные алгоритмы работы устройства с программным обеспечением должны быть максимально изучены в обстоятельствах, которые считаются внештатными для функционирования устройства. Одним из предпочтительных вариантов организации инъекции неисправностей является техника фаззинга [66, 78, 88], которая является разновидностью мутационного тестирования. Фаззинг позволяет тестировать программное обеспечение в автоматическом или полуавтоматическом режиме. Техника заключается в

передаче приложению на вход неправильных, неожиданных или случайных данных. Предметом интереса являются падения и зависания, нарушения внутренней логики и проверок в коде приложения, утечки памяти, вызванные такими данными на входе.

Совмещение данной техники с использованием аппаратных средств для диагностики отказов и сбоев является актуальным для микропроцессорных систем и требует разработки собственных инструментов отладки и контроля, в том числе специального программного обеспечения. В процессе фаззинга участвует три основных компонента: тестируемая программа, программа фаззинга, набор данных для проведения фаззинга. Набор данных представляет собой исходные данные в виде системных файлов, которые будут модифицированы в процессе работы программы фаззинга и поданы в тестируемую программу с различными видами искажений (мутаций). Программа с помощью интегрированных модулей устанавливает «зависания» и отказы программы и использует «удачные» для отказа входные данные как новые входные данные для мутации. Фаззинг включает несколько основных модулей [106]:

1. Модуль генерации данных;
2. Модуль алгоритма управления фаззингом;
3. Модуль внесения ошибок;
4. Модуль диагностики ошибок.

В зависимости от типа системы, в которой используется фаззинг, представленные модули могут быть расположены на отдельном устройстве тестирования или интегрированы совместно с эксплуатируемой системой.

Готовые решения для фаззинга программного обеспечения доступны для платформ общего назначения и воспроизводительных устройств на базе микропроцессоров. Для применения фаззинга для встраиваемых систем требуется перепроектирование и интеграция модулей. Рассмотрим классификацию средств организации тестирования на основе фаззинга. Для проведения таких испытаний на сегодняшний день не существует стандартов, только практики применения в реальных проектах крупных корпораций, таких как Google, Facebook, Microsoft

[114]. Применимость фаззинга распространяется на все виды автоматизируемых программных тестов (модульные, интеграционные, системные, графические). Особенность применения заключается в предоставлении программам фаззинга точки входа в целевую систему или программу, что организуется на этапе проработки архитектуры проектируемого приложения [39, 78, 111, 114].


Рисунок 1.8 – Классификация средств фаззинга

Одной из тенденций современной разработки программного обеспечения является исследование программных средств на потенциальные уязвимости с точки зрения информационной безопасности, а также для учета возможности физической перезаписи программ или воздействия на устройства по питанию, устойчивости к электромагнитным атакам используются специальные аппаратные устройства, контролирующие механизмы безопасности функционирования программ при изменении условий функционирования аппаратной среды.

Поскольку при разработке доверенных программных средств микропроцессорных устройств проектирование осуществляется с помощью средств компиляции, библиотек интерфейсов и протоколов, которые представляют собой огромную кодовую базу, то аттестация и верификация кода для снижения риска проявления уязвимостей является тенденциозной задачей.

Для верификации свойств программ и проверки корректности функционирования используемых библиотек, которые применены при разработке

программ микропроцессорных устройств используется современные средства статического анализа, позволяющие выявить потенциальные уязвимости кода в процессе его проектирования. Поскольку верификация кода представляет отдельную сложную задачу, то экономическая целесообразность проведения верификации для микропроцессорных устройств нецелесообразна. Поэтому рассматривают различные другие исследования программ, в том числе с применением модели «черный ящик» (таблица 1.2) [114].

Таблица 1.2 – Модели тестирования программы

| ***Критерий*** | «Белый ящик» | «Серый ящик» | «Черный ящик» |
|---|---|---|---|
| Уровень применимости | Модульное тестирование | Интеграционное Тестирование | Приемочное тестирование |
| Ответственность | Разработчик | Разработчик | Независимый тестировщик / Разработчик |
| Знание реализации | Обязательно | Обязательно для Интерфейсов | Не требуется |
| Знание сценария использования | Необходимо | Необходимо | Не известно |
| Материал для сценариев тестов | Код для всей системы | Код для интерфейсов и отдельных модулей | Спецификации |

Для непроверенных библиотек, драйверов и протоколов формируется алгоритм тестирования по методике «серого» или «черного» ящика. Это означает, что свойства библиотек считаются неизученными и исследуются в интересующих состояниях выполнения программы. Формируемая диаграмма перехода состояний позволяет оценить корректность работы тестируемых модулей. Тем не менее, разработка программ микропроцессорных устройств без учета свойств программы в части базовых связей программных компонентов и модулей приводит к повышению сложности семантических и синтаксических свойств программы, что в свою очередь приводит к риску появления ошибок. Для снижения уровня сложности программ используется проектная архитектура, которая представляет

организацию системы и описывает связи между компонентами этой системы (а также внешней средой). Архитектура позволяет определяет принципы проектирования и развития программы [107, 117, 118]. Тестирование микропроцессорной программы представляет процесс исследования, испытания, имеющий своей целью проверку соответствия между реальным и ожидаемым поведением программы на наборе тестов, разработанных с учетом требований технического задания и алгоритмов работы.

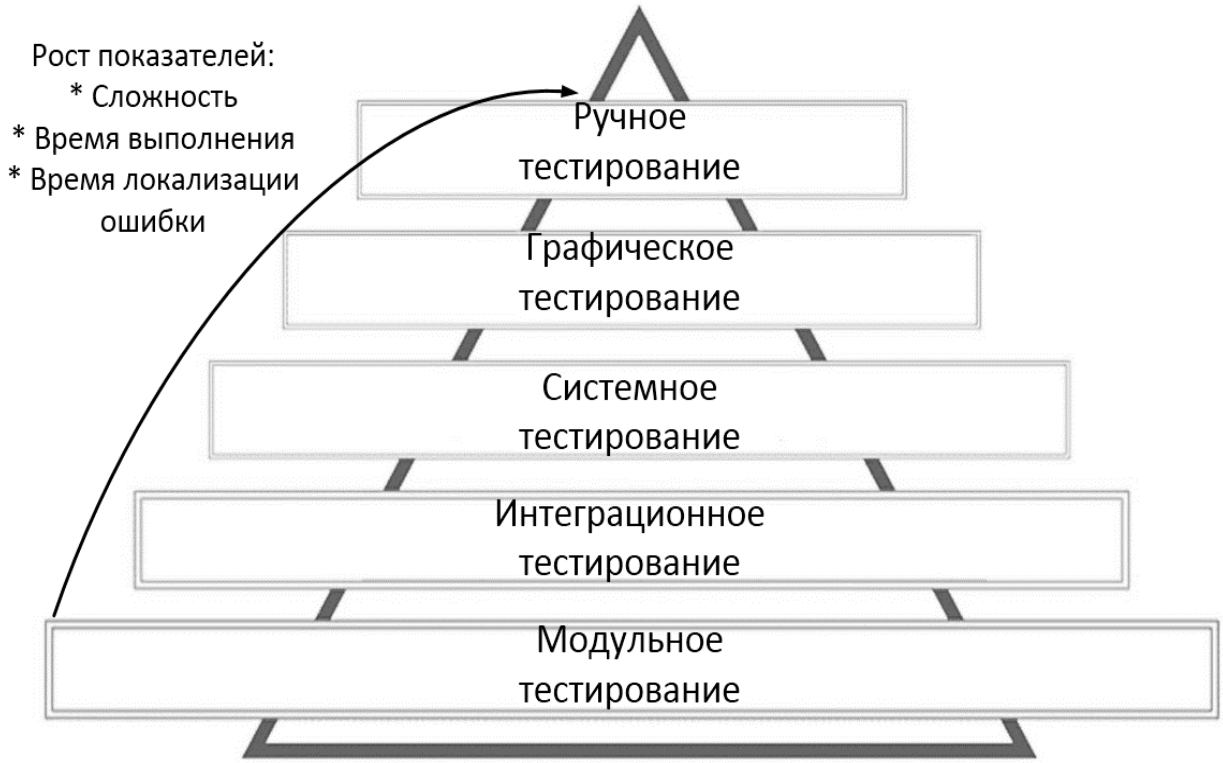



Рисунок 1.9 – Виды тестирования

Для микропроцессорных устройств проводятся все типы тестирования, при этом организация инфраструктуры тестирования является задачей, требующей не только инженерного, но и научного подхода. Тестирование в зависимости от цикла проектирования программы использует различные виды. Ресурсная стоимость и время проведения тестирования определяется в зависимости от сферы применения программного продукта.

## 1.5 Выводы по главе

1. В результате анализа работ, посвященных вопросам устойчивости программно-технических комплексов к отказам и сбоям, выявлено, что известные методики обнаружения неисправностей на основе регламентируемых методик испытаний устройств недостаточно эффективны и требуют усовершенствования. При этом существующие отраслевые стандарты указывают на необходимость осуществления имитации неисправностей при разработке (и в процессе макетирования) микропроцессорных устройств, но не определяют конкретных алгоритмов и необходимый комплекс технических средств для проведения испытаний.

2. Установлено, что проектирование уникальных и малосерийных устройств информационных комплексов, имеющих сложноопределяемые или неизвестные характеристики компонентов, сопряжено со сложностями расчетов традиционных показателей надежности, поэтому нуждается в специально разработанном оборудовании для проведения экспериментов для выявления дефектов программных и аппаратных средств.

3. Определено, что отсутствует гармонизация между отечественными и зарубежными стандартами в области проектирования цифровых устройств, включая анализ видов и последствий отказов. Это затрудняет процесс проектирования и установления характеристик работоспособности устройств в связи с необходимостью замены качественных оценок безопасности и надежности при работе устройства в реальных условиях (используемых в отечественных стандартах) на количественные показатели (используемые зарубежными стандартами). Выходом из сложившейся ситуации является применение экспериментальных исследований, включающих имитацию неисправностей для оценки работоспособности программ и аппаратуры совместно.

4. Установлено, что существующие имитаторы неисправностей для исследования и обеспечения работоспособности цифровых информационных систем, представленные на коммерческих рынках, являются оборудованием,

внутреннее устройство которого является защищаемой интеллектуальной собственностью. В связи с этим возникает необходимость создания собственных средств для имитации неисправностей и анализа последующей реакции микропроцессорных устройств, что при проектировании является единственным вариантом для создания отказоустойчивых систем.

5. Установлено, что в связи с тем, что используемые микропроцессорные элементы аппаратных средств относятся к классу высоконадёжных изделий, то основной проблемой обеспечения работоспособности проектируемых устройств является выявление отказов, связанных с дефектами программного обеспечения, проявляемыми в реальной аппаратной среде. Для выявления этих дефектов представляется целесообразным разработка имитаторов неисправностей, которые позволяют исследовать корректность работы всех программных функций в режимах эксплуатации устройства.

6. Определена актуальность исследования, которая обусловлена возрастающими требованиями к качеству и надежности программно-аппаратных комплексов, а также отсутствием практических рекомендаций и недостаточной эффективности существующих методов и средств для проектирования и отладки микроконтроллерных устройств. Установлена целесообразность разработки алгоритмов и технических средств, которые обеспечивают выявление программных и аппаратных дефектов в информационных системах и могут быть применены в различных областях отечественной промышленности.

# 2 РАЗРАБОТКА ПРОГРАММНОГО БЛОКА ИМИТАЦИИ ОТКАЗОВ И СБОЕВ С ПОМОЩЬЮ ИСКАЖЕНИЯ ВХОДНЫХ ДАННЫХ МИКРОПРОЦЕССОРНОГО УСТРОЙСТВА

Вторая глава **посвящена** описанию устройства имитации отказов и сбоев. Рассмотрен состав программного блока для имитации неисправностей исследуемого устройства и способ имитации на основе искажения входных данных. Рассмотрена интеграция устройства в информационный комплекс и алгоритм диагностики для передачи информации о найденных неисправностях.

## 2.1 Выбор и обоснование режимов искажения информации для имитации отказов и сбоев микропроцессорного устройства

Для обоснования критериев и выбора режимов имитации неисправностей необходим анализ имитаторов неисправностей (таблицы 2.1 - 2.2). Имитаторы неисправностей возможно классифицировать по ряду признаков, представленных в таблице. Эффективность испытаний характеризуется сокращением времени их проведения, возможностью всесторонних проверок и получением наиболее достоверных результатов.

Таблица 2.1 – Классификация устройств имитации неисправностей

<table>
<tr><th colspan="2">1. Назначение исследования</th><th rowspan="2">2. Тип имитируемой неисправности</th><th rowspan="2">3. Степень адекватности оценки</th></tr>
<tr><th>объект</th><th>цель</th></tr>
<tr><td>Поколение</td><td rowspan="7">FMEA<br>виды и последствия отказов<br>FMECA<br>эффектов режима отказа и критичности<br>FMEDA<br>диагностика отказов<br>Синтез и отладка тестов<br>Комбинированные</td><td>Постепенные отказы</td><td rowspan="7">Точность оценки (установлена/ не установлена)<br><br>Способ оценки (эксперимент/ модель)</td></tr>
<tr><td>Сфера использования</td><td>Постоянные отказы</td></tr>
<tr><td>Степень сложности</td><td>Перемежающиеся отказы</td></tr>
<tr><td>Принцип действия</td><td>Связанные отказы</td></tr>
<tr><td>Конструкция/ исполнение</td><td>Сбои</td></tr>
<tr><td>Элементная база</td><td>Комбинированные отказы и сбои</td></tr>
</table>

Таблица 2.2 – Классификация устройств имитации неисправностей (продолжение)

<table>
<tr><th>4. Тип воздействия</th><th>5. Тип искажения</th><th>6. Уровень имитации последствий</th></tr>
<tr><td>Электрические сигналы</td><td rowspan="6">Случайные<br><br>Регулируемые аналоговые/дискретные<br><br>Регулируемые цифровые<br><br>Комбинированные</td><td>Транзисторные</td></tr>
<tr><td>Электромагнитное воздействие</td><td>Вентильные</td></tr>
<tr><td>Лазерное излучение</td><td>Малые/средние/ сверхбольшие интегральные схемы</td></tr>
<tr><td>Рентгеновское излучение</td><td>Типовой элемент замены</td></tr>
<tr><td>Альфа-, бета- и гамма-излучение</td><td rowspan="2">Устройство/блок/ подсистема</td></tr>
<tr><td>Программное искажение</td></tr>
</table>

Достоверность результатов, получаемых в ходе испытаний, является наиболее важным критерием при выборе средств имитации и может быть обеспечена при адекватности процесса испытаний процессу эксплуатации объекта испытаний (ОИ). Обеспечение адекватности можно рассматривать с нескольких точек зрения:

– адекватность условий испытаний условиям реальной работы ОИ;

– адекватность имитируемых последствий неисправностей реальным, возникающим в ОИ в процессе эксплуатации;

– адекватность состояний ОИ в момент имитации неисправностей состояниям наиболее вероятным при реальной эксплуатации.

Для классификации способов имитации неисправностей представим работу микропроцессорного устройства следующим образом:

$$Z = \Psi(X, Y_n, T), \tag{2.1}$$

где $\Psi$ ($\Psi$*) – функция устройства, выполняемая корректно (некорректно);

$Z$ ($Z$*) – вектор-шаблон результатов выполнения функции (при искажении);

$X$ – вектор входных данных, обрабатываемых устройством; $Y_n$ – вектор переменных состояния устройства, характеризуемых с помощью значений, хранимых в

регистрах команд и оперативной памяти программы устройства; $T$ – вектор времени выполнения функций устройства.

Вектор-шаблон результатов выполнения функции формируется при работе устройства в штатном режиме с контролем корректности выполнения. Полученные значения сравниваются с реальным исполнением функций устройства при его тестировании. При работе устройства в период возникновения отказов и сбоев функция устройства и вектор-шаблон результатов не совпадают (при условии корректно полученных значений вектора-шаблона).

Для внесения искажений возможно использовать программные и аппаратные средства. Аппаратные средства представляют технические системы (контактные и бесконтактные), 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 $\Psi$ исправного устройства позволяет определить тип используемого имитатора неисправностей, который удовлетворяет требованиям к стоимости и направленности поиска в проектируемой системе.

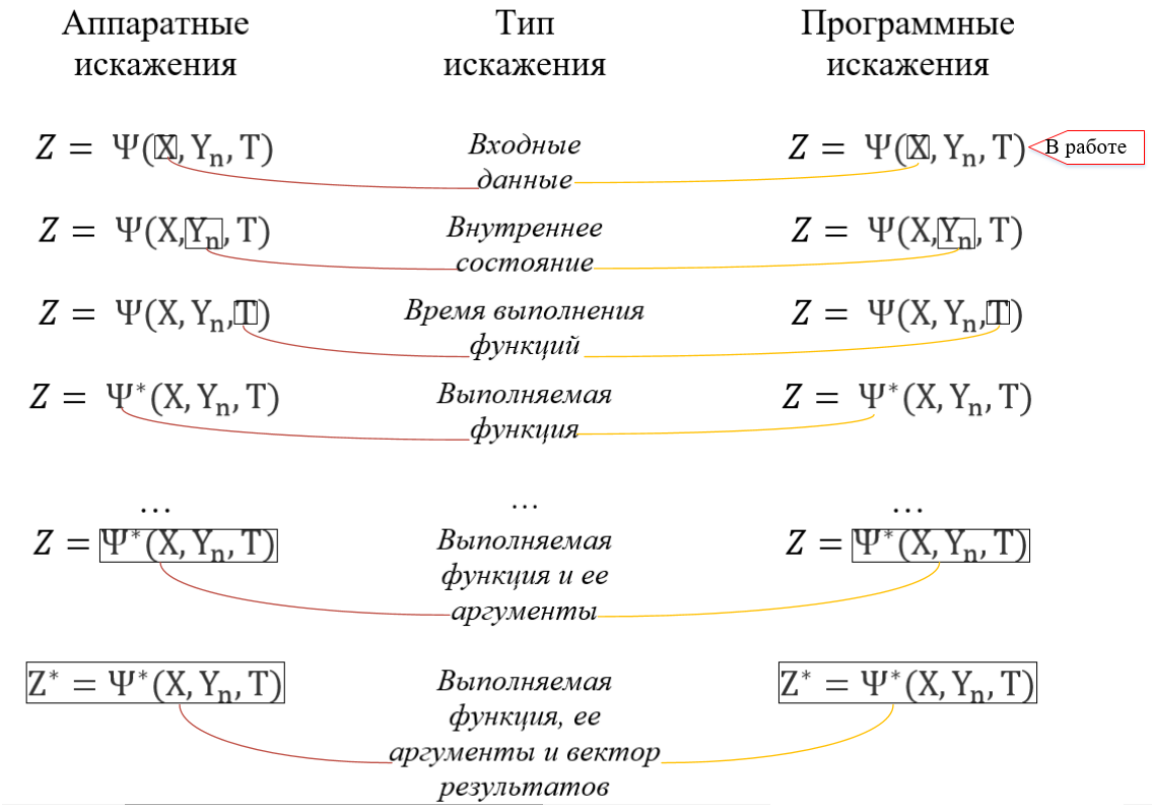



Рисунок 2.1 – Способы имитации неисправностей

Для каждого компонента выбран вариант реализации, который позволяет осуществить имитацию на заданном уровне детализации [20, 86]. Рассмотрим описание конфигуратора для выбора параметров имитации неисправностей. Наиболее распространёнными способами имитации неисправностей являются искажение входных данных (вектор $X$), состояния функционирования устройства (вектор $Y_n$), времени выполнения функций ($T$). Более сложным в реализации и контроле является искажение с помощью выполняемой функции $\Psi$ и (или) одного или нескольких перечисленных аргументов ($X$, $Y_n$, $T$).

Современные микропроцессоры, используемые при разработке цифровых устройств, относятся к классу высоконадёжных изделий. Показатели безотказности микропроцессоров очень высоки (интенсивность отказов составляет один отказ на сто миллионов приборочасов). Для невосстанавливаемых устройств наработка на отказ $T$ определяется следующим образом [5, 6]:

$$T = \exp \int_0^{\infty} p(t)dt, \tag{2.2}$$

где $p(t)$ – вероятность безотказной работы; $t$ – время работы устройства.

Тем не менее, вычисление наработки на отказ для устройств с собственным ПО не предусмотрено. Проблема надёжности ПО и объединение микропроцессорных модулей с набором программ в единую систему увеличивают вероятность проявления свойства эмерджентности, что приводит к неожидаемым отказам и сбоям [13, 14, 28].

Для обеспечения работоспособности устройства отклонение между выходной функцией и функцией-шаблоном должно быть минимально для выполнения основных функций в режимах работы микропроцессорного устройства:

$$\sum_{i=1}^{M} [Z_i - Z_i^*] \to \min, \tag{2.3}$$

где $M$ – число сравнений идеальной и реальной функции за время проведения тестирования. Если рассмотреть уравнение $Z$ как уравнение от одной переменной, то другие аргументы будут представлять набор параметров. Для определения этих параметров требуется решить систему уравнений:

$$\begin{cases} \sum_{i=1}^{M}[Z_i - \Psi(x_i, y_n, t)](\frac{d\Psi}{dy_n})_i = 0, \\ \sum_{i=1}^{M}[Z_i - \Psi(x_i, y_n, t)](\frac{d\Psi}{dt})_i = 0. \end{cases} \quad (2.4)$$

Решить систему уравнений в общей форме нельзя [6] (для этого необходимо знать конкретный вид функции $\Psi$), поэтому требуется проведение экспериментальных исследований. При проявлении эмерджентных свойств при работе устройства вид функции $\Psi$ может меняться [34, 65], т.е. она имеет сложноопределяемый вид, зависящий от времени. Непостоянство вида функции $\Psi$ также связано со спецификой применяемых микросхем и способствует проявлению отказов и сбоев. К примеру, исследование состояний регистров для микропроцессорного модуля Atmega128 показало наличие недокументированных функций микропроцессора [27], таких как имитация «зависания», несанкционированная передача данных, изменение во времени характеристик микросхем с выходом за пределы, указанные в технической документации.

Все имитаторы неисправностей можно классифицировать с помощью предложенных ранее способов искажения для проектируемой системы тестирования информационного комплекса [79, 81, 85].

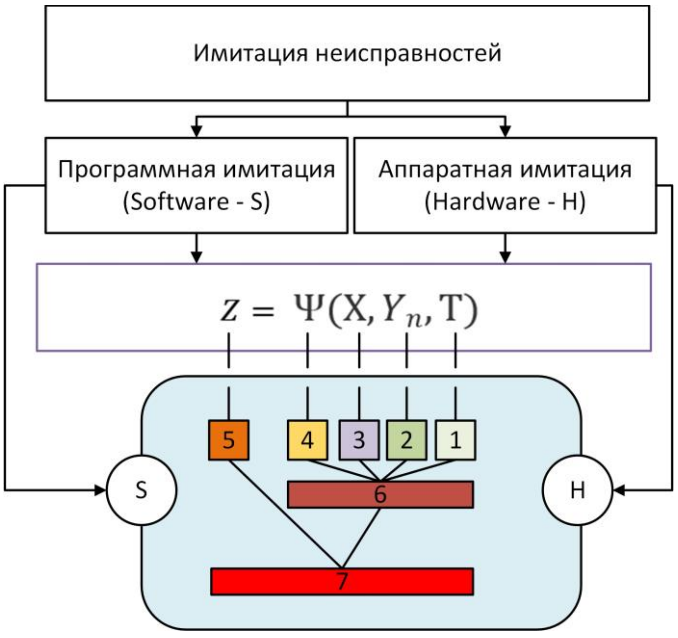



Рисунок 2.2 – Общая структура имитатора неисправностей

Поскольку эксплуатация устройства производится для информационного комплекса, то требуется учитывать особенности выбора имитатора неисправностей для совместного использования в составе комплекса. Для определения причины отказа или сбоя устройств программно-аппаратного комплекса из-за высокой степени связности может быть затруднена. На данный момент не существует единой системы классификации неисправностей для комплекса, применимой для поиска отказов и сбоев информационных систем [33] и определения отказоустойчивости отдельных узлов (микропроцессорных устройств).

Поскольку комплекс содержит множество устройств и существуют вероятностные события, которые могут быть причиной ложных отказов и сбоев, то при диагностике комплекса учитывают все известные события [30, 40, 45], формируя показатель энтропии совместных событий:

$$G(X) = -\sum_{i=1}^{N} p_i \sum_{j=1}^{M} p_j \log_2 p_i(j), \qquad (2.5)$$

где $i$ – состояние, зависящее от предыдущего выбора узла; $p_i$ – вероятность оптимального значения параметра; $p_i(j)$ – вероятность выбора $j$-го при условии предыдущего выбора $i$-го узла.

Существующие классификации используют либо описание модели программных дефектов проектируемой системы, либо опираются на введение неисправностей путем модификации памяти испытуемого устройства. Классификация по внедрению неисправностей не рассматривает возникновение полноценных отказов и использует лишь инструмент модификации памяти или гибридные инструменты, которые эффективны для специализированных архитектур устройств, но не адаптированы под микропроцессорные устройства. Таким образом, программные и аппаратные сбои/отказы практически не пересекаются в системах моделирования и рассматриваются на отдельных уровнях, что значительно снижает надежность системы и затрудняет диагностику [70].

Поэтому для проектирования устойчивых к отказам и сбоям микропроцессорных устройств в составе комплекса требуется обеспечить эффективность использования имитации неисправностей [115, 118], при которой

учитываются значимые причины возникновения отказов и сбоев. Наиболее простой способ имитации представляет использование искажения входных данных, поскольку информационный комплекс реализует процедуры приема и передачи и максимально уязвим именно в процессе обмена информацией. Принятая классификация типов неисправностей позволяет определить меры, необходимые для устранения каждого вида отказа, поскольку часть сбоев не вносит в систему критических изменений (рис. 2.3).

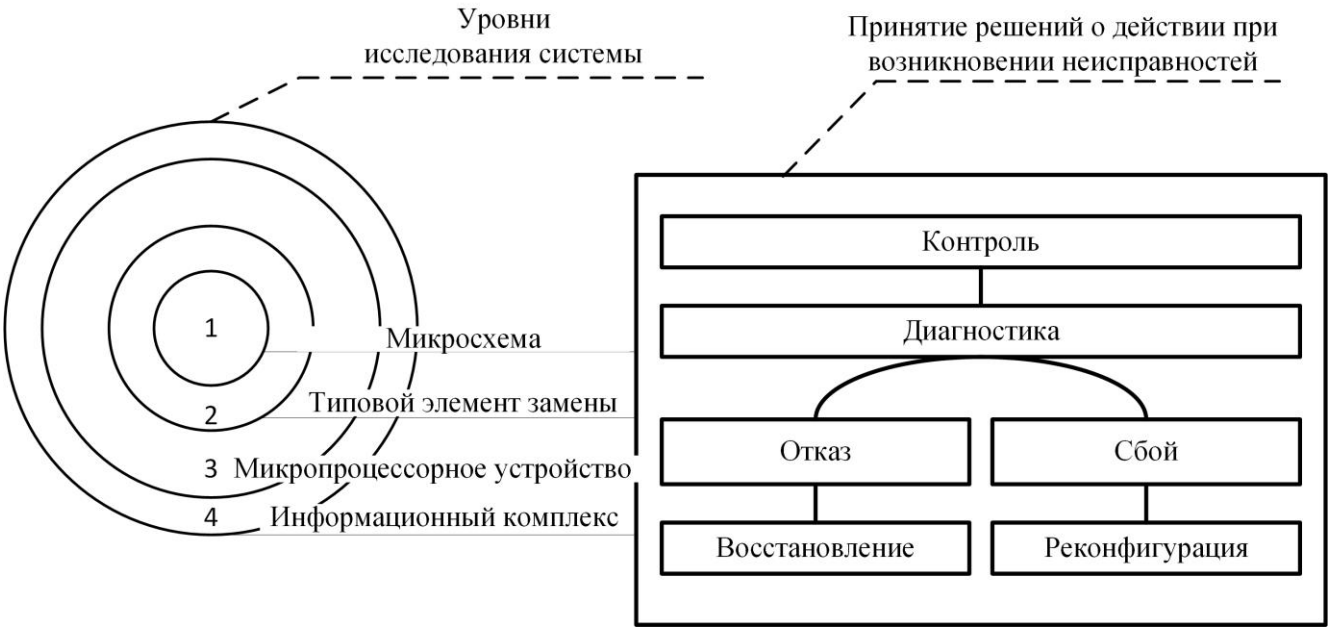



Рисунок 2.3 – Принятие решений для поиска и устранения отказов и сбоев

Иерархическое описание проектируемой системы можно представить в виде нескольких уровней – на уровне микросхем, типовых элементов замены, микропроцессорного устройства или информационного комплекса. На всех уровнях описания действуют аппаратно-программные механизмы контроля, диагностирования, идентификации, реконфигурации, а также восстановления вычислительного процесса. Сложность анализа уже разработанного устройства заключается в том, что разработчик может не иметь полной и достоверной информации о всех контурах контроля, диагностирования, идентификации, восстановления и реконфигурации на всех уровнях описания системы.

Для исправления найденных проблем предлагается использовать программные способы блокирования для выбранного типа неисправностей. Имитация используется для получения информации и предотвращения части отказов и сбоев, которые требуют наличия дополнительного программного кода в зависимости от типа исправляемого отказа или сбоя. При разработке устройств комплексной системы виды реакций выбираются исходя из требований технического задания [70]. Для реализации процедур контроля и диагностики неисправностей программно-аппаратного комплекса создана тестовая среда, позволяющая осуществлять имитацию отказов.

В соответствии с введенными для микропроцессорных устройств информационного комплекса требованиями целесообразно исследовать устойчивость к отказам и сбоям с помощью способа искажения входных данных (вектор $X$), поскольку данный вид имитации способствует обнаружению ошибок ПО и аппаратных дефектов и легко интегрируется в существующие информационные комплексы.

## 2.2 Разработка модуля инъекции неисправностей в составе программного блока имитации отказов и сбоев

Для анализа средств автоматизации испытаний на надежность используется функционально-структурный подход, в основе которого лежит идея об определяющей роли функции в диалектической взаимосвязи функции и структуры. Выбор данного подхода связан также с качеством анализируемого материала, в котором практически отсутствует описание структур средств испытаний и в то же время имеется описание функций с различной степенью детализации. Определение набора функций позволит в дальнейшем разработать новые средства для их реализации, многофункциональные либо специализированные, с учетом всех требований, предъявляемых к процессу автоматизации испытаний [2, 25, 92]. Функциональная организация системы представляет собой модель системы, построенную на основе функциональных элементов и отражающую основные

функциональные связи между ними. Основу описания функционально-структурной организации системы составляет дерево функций, адекватное по своей структуре иерархической организации системы и раскрывающее многоуровневое представление функций системы. Процесс анализа систем включает выявление основных и дополнительных функций систем-прототипов, построение обобщенного дерева функций, выявление базовых структур и анализ реализации функций в структурах.

*Первой* функцией, явно либо неявно присутствующей во всех прототипах имитаторов неисправностей, является функция планирования эксперимента, в соответствии с которой задаются типы неисправностей, типы отказавших компонентов в объекте испытаний и условия имитации – состояние ОИ.

*Вторая* функция предоставляет возможность синхронизации моментов имитации неисправностей к определенному состоянию устройства – этапу, фазе работы программы и обеспечивает возможность всесторонних проверок. Наличие этой функции связано с тем, что реакция микропроцессорного устройства будет зависеть от его состояния в момент имитации неисправности [3, 27, 42].

*Третья* функция имитации неисправностей является основой автоматизации испытаний на устойчивость к отказам и сбоям. Для реализации дайной функции в известных системах использовались два места подключения к ОИ: выводы микросхем и интерфейсы.

*Четвертой* функцией является функция анализ реакции устройства на проимитированную неисправность является и присутствует в описании трех систем автоматизации испытаний на устойчивость к отказам и сбоям.

*Пятая* функция представляет функцию сбора статистики по результатам экспериментов с введением неисправностей ОИ и вычисления количественных характеристик надежности по известным положениям математической статистики. Базовый состав, лежащий в основе автоматизации испытаний, включает восемь функций [1, 112]:

1) $Ф_1$ – задание типа отказавшего компонента в устройстве;

2) $Ф_2$ – задание типа неисправности отказавшего компонента;

3) $Ф_3$ – задание условий имитации неисправности состояния устройства;

4) $Ф_4$ – выделение заданного состояния устройства (синхронизация);

5) $Ф_5$ – имитация неисправности компонента в устройстве;

6) $Ф_6$ – анализ реакции устройства на проимитированную неисправность;

7) $Ф_7$ – набор статистики по результатам экспериментов;

8) $Ф_8$ – вычисление количественных характеристик надежности устройства.

В проанализированных источниках [72, 74, 81, 86] существует, по крайней мере, два подхода к процессу имитации. Первый заключается в том, что имитируется сама неисправность, например, короткое замыкание выхода (входа) микросхемы на землю. Этот подход очень ограничен в связи с ростом степени интеграции компонентов и ограничения доступа к ним.

Второй подход основан на имитации последствий неисправностей – ошибок, которые возникают вследствие изменения структуры компонентов отказов или сбоев. При реализации второго подхода функция синхронизации дополняется выделением условий проявления неисправности в месте имитации, так как неисправность может быть потенциальной (если компонент не участвует в выполнении функции) и действующей, когда работа неисправного компонента приводит к появлению ошибки, которая и имитируется в устройстве.

Условия проявления неисправности фактически определяют момент перехода неисправности из потенциальной в действующую. Реализация второго подхода и имитации неисправностей приводит к необходимости определения условий проявления неисправностей и разработке моделей ошибок – последствий неисправностей компонентов ОИ, которые имитируются в функции $Ф_5$. Базовый состав функций, лежащий в основе второго подхода, должен включать:

1) $Ф_1$ – задание типа отказавшего компонента в устройстве;

2) $Ф_2$ – задание типа неисправности отказавшего компонента;

3) $Ф_{2.1}$ – определение последствий неисправности компонента в месте имитации для определения ошибки;

4) $Ф_{2.2}$ – определение условий проявления неисправности в месте имитации;

5) $Ф_3$ – задание условий имитации неисправности, то есть состояния устройства;

6) $Ф_4$ – выделение заданного состояния устройства (синхронизация);

7) $Ф_5$ – выделение условий проявления неисправностей компонента в точке имитации неисправностей;

8) $Ф_5$ – имитация последствий неисправности;

9) $Ф_6$ – анализ реакции ОИ на проимитированную неисправность;

10) $Ф_7$ – набор статистики по результатам экспериментов;

11) $Ф_8$ – вычисление количественных характеристик надежности устройства.

Обобщенные структуры, отражающие взаимосвязи функциональных компонентов в процессе исследования, представлены соответственно для первого и второго вариантов имитации неисправностей. Функции $Ф_3$ - $Ф_5$ составляют основу автоматизации испытаний, реализация $Ф_5$ представляет основную задачу настоящей работы, поэтому дальнейшую декомпозицию проведем для этих функций.

Процесс выделения определенного состояния устройства связан со сбором информации о системе и сравнением собранной информации с определенным эталоном, информационным портретом в момент имитации неисправности, поэтому функция $Ф_4$ может быть разделена на две:

1) $Ф_{4.1}$ – сбор информации о состоянии аппаратурных и программных компонентов устройства;

2) $Ф_{4.2}$ – сравнение собранной информации с заданным эталоном.

При реализации функции имитации по второму варианту возможна декомпозиция $Ф_5$ на следующие составляющие:

1) $Ф_{5.1}$ – сбор информации о состоянии компонента, неисправность которого имитируется;

2) $Ф_{5.2}$ – сравнение собранной информации с эталоном, определяющим переход неисправности из потенциальной в действующую.

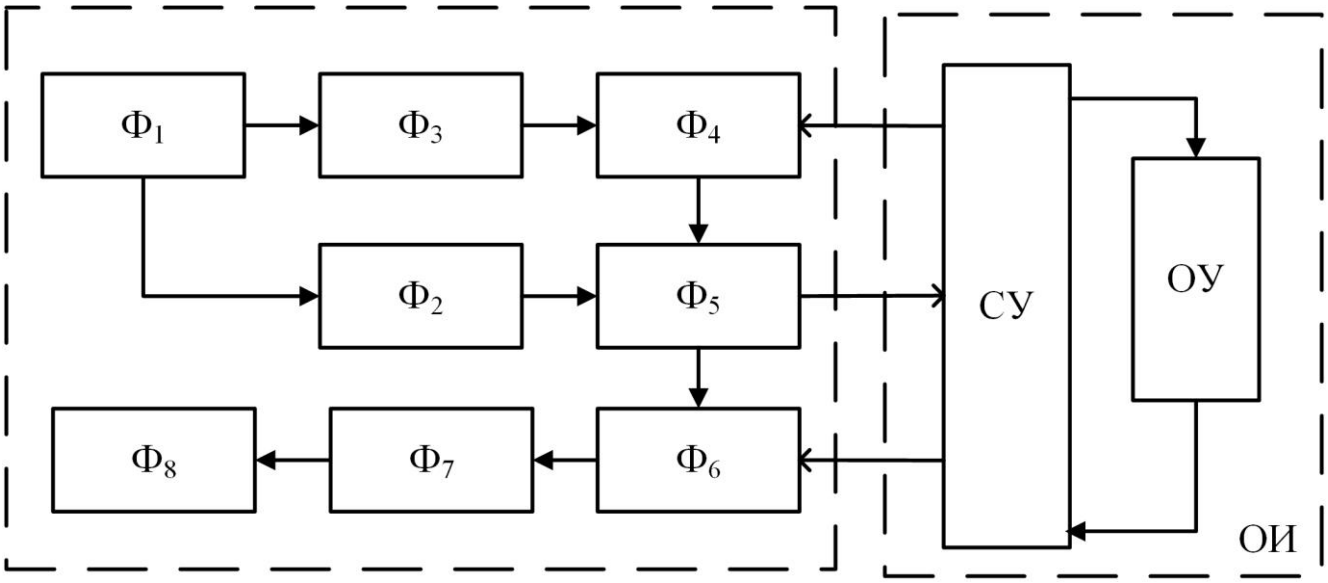



Рисунок 2.4 – Модель АСИ на основе имитации неисправностей

Имитация последствий неисправности $Ф_5$ заключается в искажении сигналов в системе, эквивалентном последствиям неисправности (имитация ошибки). Функция анализа реакции устройства во многом аналогична функции синхронизации, разница состоит лишь в том, что при синхронизации необходимо выделить одно состояние ОИ, а при анализе реакции выделяются определенные множества состояний ОИ, свидетельствующие о его отказе либо сбое. В связи с этим возникает задача сжатия исходного объема информации, которую предлагается решать тремя способами: предварительным выделением критериев отказа ОИ и регистрацией в ОИ небольшого объема информации; сжатием информации о состоянии ОИ и последующим ее анализом, комбинацией первых двух.

Каждый из этих способов имеет свои преимущества и недостатки. Декомпозиция $Ф_6$ может быть проведена следующим образом:

1) $Ф_{6.1}$ – сбор информации о состоянии аппаратурных и программных компонентах устройства;

2) $Ф_{6.2}$ – сжатие исходного информационного массива;

3) $Ф_{6.3}$ – определение отказов и сбоев устройства.

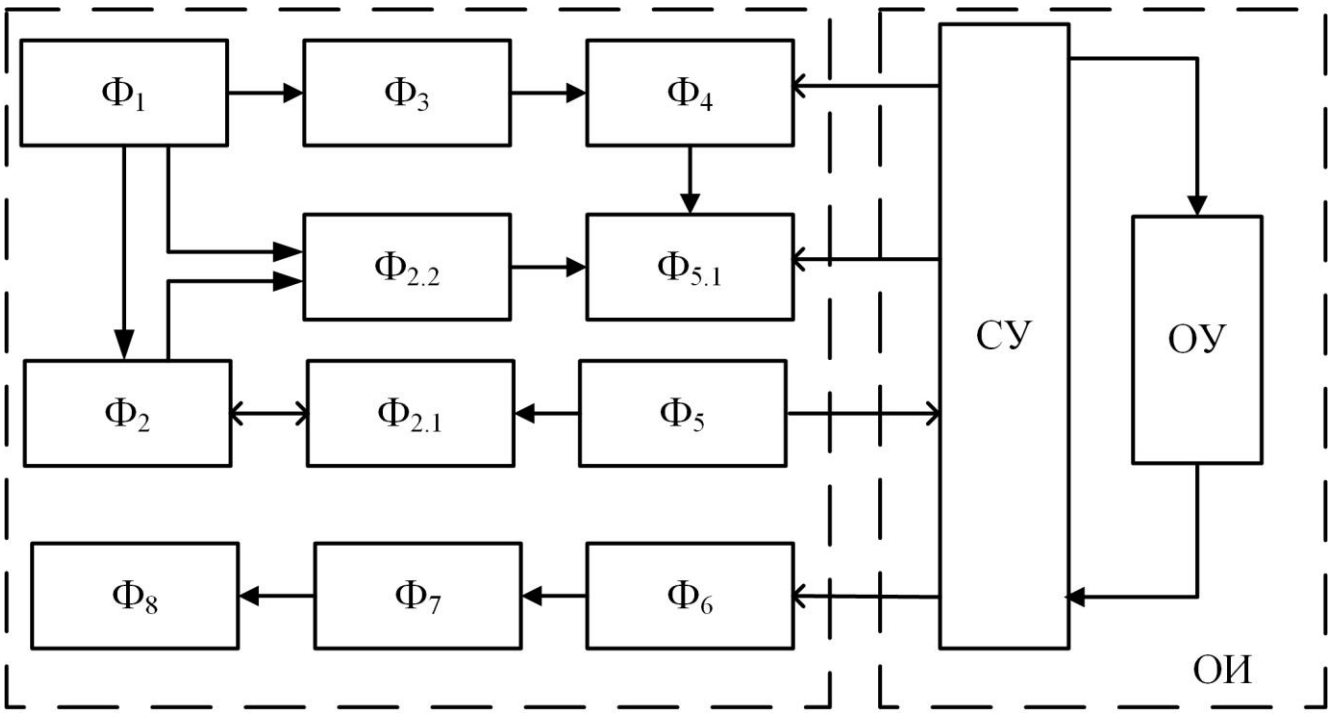



Рисунок 2.5 – Модель АСИ на основе имитации последствий неисправностей

Вопросы реализации других функций, обеспечивающих процесс автоматизации испытаний, будут рассматриваться только в той мере, в какой это необходимо для анализа и синтеза средств имитации неисправностей. Проведенный анализ позволил выделить набор функций, лежащий в основе автоматизации испытаний, на устойчивость к отказам и сбоям и детально рассмотреть функции, возлагавшиеся на средства имитации неисправностей. Имитация отказов и сбоев для микроконтроллерного устройства обладает спецификой, связанной с выбором необходимых компонент среди всех методов и средств, используемых для проектирования устройств с микропроцессорными модулями, поэтому комплекс мер основан на применении набора инструментов, которые можно выделить в специализированную архитектуру. Выделим основные уровни имитации неисправностей [32, 76]:

– имитация на уровне интерфейса – воздействие на шину данных микроконтроллера, обусловленное потерями и искажениями в получаемой информации или воздействием на процессор;

– имитация на уровне процессора – выполнение инструкций процессора, являющихся опасными для надежной работы;

– имитация на уровне программы – выполнение программ, которые могут привести к отказам процессора и интерфейсов, логическим и аппаратным отказам;

– имитация на уровне ввода-вывода – подача непредусмотренных данных алгоритмами обработки ввода-вывода.

Для устройства предлагается следующий вариант архитектуры автоматизированного тестирования на отказы и сбои, который соответствует стандартам IEC 61508 и EN 50128 [119]: архитектура состоит из двух подсистем – объекта управления (испытуемый блок) и управляющей системы (станции управления - СУ), нацеленной на имитацию и подготовку результатов поиска неисправностей. На рисунке 2.6 представлена функциональная схема для оценки последствий имитации отказов и сбоев в исследуемой системе.

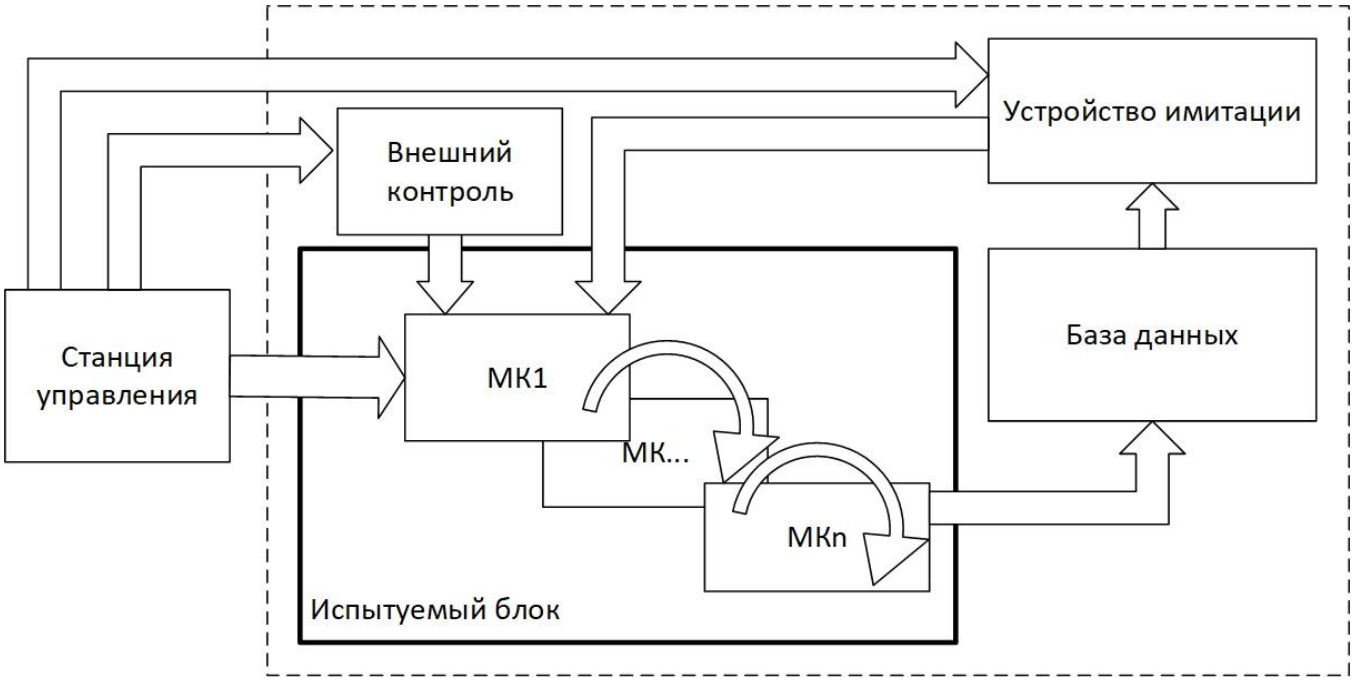



Рисунок 2.6 – Архитектура системы автоматизированного тестирования

Испытуемый блок представляет собой совокупность тестируемых микропроцессоров с общим входом и общим выходом (их количество равно $N$). На них исполняется тестовая программа, которая обрабатывает входную последовательность данных и выполняет ряд действий, требуемый для

обнаружения неисправностей в рамках теста. Данные о результатах снимаются с помощью устройства внешнего контроля и записываются в базу данных (БД).

Блок устройства имитации (УИ) представляет собой устройство с работающей программой по классификации признаков сигнатур, выбирающей данные с блока СУ и сохраняющей на этом блоке информацию о поданной на выход блока последовательности с меткого времени.

Блок внешнего контроля осуществляет контроль частей испытуемого блока, список которых корректируется с блока станции имитации и связан с блоком УИ, которое по командам с СУ (реализованном в виде ПК) осуществляет запуск тестовых последовательностей. СИ представляет собой станцию имитации, которая организует установку сигналов внутри испытуемого блока.

Отдельно следует рассмотреть технические средства имитации неисправностей. В качестве такого блока используется аппаратные устройства с поддержкой JTAG-интерфейса, которые являются стандартными для большинства микроконтроллеров архитектур [50, 61, 63] *DSP, ARM, AVR, MIPS.* Классическое тестирование системы согласно по классификации ГОСТа Р 51904-2002 «Программное обеспечение встроенных систем» осуществляется для поиска ошибок программного и аппаратного обеспечения совместно.

Интерфейс JTAG, стандартный для микропроцессорных устройств позволяет получить доступ к процессору и интерфейсам заданного микроконтроллера. Для выполнения мероприятий по тестированию, описанных в стандарте IEC 61508 алгоритмы инъекции могут быть реализованы в рамках представленной системы на станции управления и устройстве имитации.

В данной работе проведение испытаний производится с использованием программного блока имитации неисправностей с помощью специально спроектированного устройства имитации неисправностей, основанного на использовании средств отладки микропроцессорных систем (диагностика по интерфейсу JTAG и анализ реакции).

Таблица 2.3 – Организация классического тестирования системы

| Тип ядра | Вид теста | Способ организации теста на процессоре |
|---|---|---|
| *ARM*-,*MIPS*-процессоры | Модульный | Проверка автомата состояний контроллера и отдельных функций путем генерации входных значений через удаленный интерфейс ПК |
| | Интеграционный | Имитация сигналов интерфейсов с помощью FPGA в режиме трансляции и перехвата сигналов системы |
| | Стресс-тест | Запуск процедур генерации сигналов на FPGA с увеличением скорости работы до фиксации отказа алгоритма |
| *DSP*-процессоры | Модульный | Составление дерева вызов функций и списка решений, полное покрытие отчетами вызовов с передачей на ПК. Анализ отчетов на ПК с помощью специальных алгоритмов |
| | Интеграционный | Взаимодействие с САПР на ПК через общий буфер обмена для генерации корреляционной функции вычисляемых сигналов |
| | Стресс-тест | Имитация сигналов с помощью построения экосистемы на генераторе сигналов, управляемого сервером непрерывной интеграции |
| *FPGA* | Модульный | Верификация с помощью встроенных средств «testBench» |
| | Интеграционный | Проверка вместе с процессорами ARM, DSP и их симуляцией |
| | Стресс-тест | Управляемая реконфигурация алгоритмов FPGA с возможностью вывода результатов работы на ПК |

Поскольку представление в стандартах технической реализации тестирования не приведено, то разработчикам приходится использовать собственные наработки и технические возможности проектируемых микроконтроллеров, микропроцессоров и микросхем для устройств [46,48].

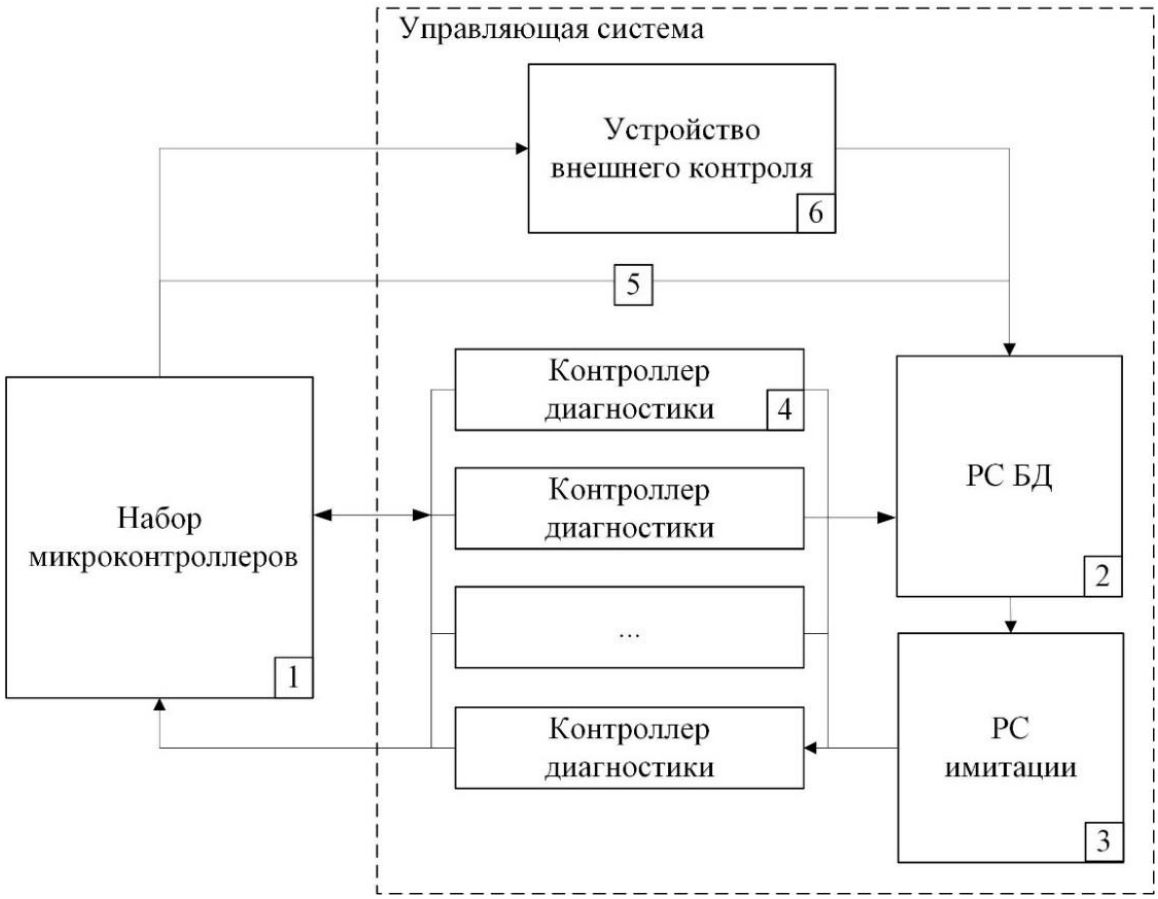



Рисунок 2.7 – Устройство имитации неисправностей и анализа реакции

Устройство описывает процесс проведения испытаний устройства с непрерывно выполняемым программным обеспечением, осуществляемый в автоматическом режиме. Применение устройства позволяет сократить время обнаружения неисправностей, автоматизировать процесс их поиска за счет анализа регистров и ОЗУ на наличие отклонений. Управляющая система осуществляет эксплуатацию программно-аппаратного комплекса (тестируемая система).

Для моделирования состояний устройств используются гибридные модели, в которых гибридные автоматы рассматриваются как особый класс динамических систем с непрерывным временем, что вызвано прежде всего совместным функционированием непрерывных и дискретных объектов и характерно для цифровых систем управления, в которых присутствует непрерывный объект управления и дискретное устройство управления (контроллер). Гибридным автоматом в современной интерпретации принято называть совокупность [2, 12]:

$$H = \{S, X, f, Start, Inv, E, G, R\}, \quad (2.6)$$

где $Y_n = \{y_{n_1}, y_{n_2}, ...y_{n_n}\}$ – множество дискретных состояний автомата или режимов работы; $X = R^n$ – множество непрерывных переменных; $f : Y_n \times X \to X$ – вектор функции, являющейся правой частью системы обыкновенных дифференциальных уравнений первого порядка относительно $x \in X$, зависящей от дискретных состояний автомата; $Start \subseteq Y_n \times X$ – множество начальных состояний; $Inv : Y_n \to P(X)$ – множество инвариантов, условий, выполняющихся в дискретных состояниях; $E \subseteq Q \times Q$ – множество переходов из одного дискретного состояния в другое; $G : E \to P(X)$ – условия перехода; $R : E \to P(X \times X)$ – отношения сброса, множество правил переопределения начальных условий, заданных для каждого перехода $e \in E$ для непрерывных переменных $x \in X$.

Для исследования графа программы используется матрица смежности, которая позволяет понять плотность сети *A*, которая характеризует связность переходов между состояниями программы [5, 18]

$$A = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{\substack{j=1, \\ j \neq i}}^{N} x_{ij}, \quad (2.7)$$

где $x_{ij}$ – бинарное отношение *i*-го и *j*-го узлов матрицы смежности графа; $x_{ij} = \{0,1\}$; $N$ – число узлов в графе;

Временная сложность *O* определяется для исследования алгоритма пути программы, состоящего из набора базовых блоков (команд ветвлений в ассемблере)

$$O(\sum_{i=1}^{C_p} C_{b_p}^{p_i} C_b), \quad (2.8)$$

где $C_b$ – количество базовых блоков; $C_p$ – количество путей выполнения; $C_{b_p}^{p_i}$ – количество точек ветвлений в *i*-м пути.

Теория гибридных автоматов и представление программы в виде графа применимо для организации программного блока инъекции в микропроцессорном устройстве. Представление программы устройства в виде направленного графа очень часто используется для диагностики его состояний и позволяет осуществлять

точечное внесение искажений данных [58, 78, 88]. Вершинами графа являются инструкции ветвлений, которые осуществляют передачу управления между функциями программы.

## 2.3 Разработка модуля обнаружения неисправностей в составе информационного комплекса

Для обнаружения неисправностей в информационном комплексе используется статистика обмена данными о функционировании в режимах штатной работы (приема и передачи) микропроцессорных устройств. Для этого используются известные алгоритмов ранжирования и обработки признаков (математической статистики и теории вероятности, деревья принятия решений, нечеткого логического вывода). Применение статистических алгоритмов позволяет классифицировать набор параметров по их важности для достижения целей обмена информацией между узлами комплекса.

Приведенные результаты анализа моделируемых отказов и сбоев, которые разделены по классам объектов, для которых по статистическим данным был обнаружен отказ или сбой. Для быстрого определения отказа или сбоя при работе микропроцессорного устройства возникает задача разработки алгоритма быстрого поиска пути между узлами, учитывающего вышеперечисленные особенности комплекса связи для передачи диагностической информации [50, 113].

Отбор признаков для формирования статистики является важным этапом для получения автоматического выявления признаков возникновения в микропроцессорных устройствах отказов и сбоев.

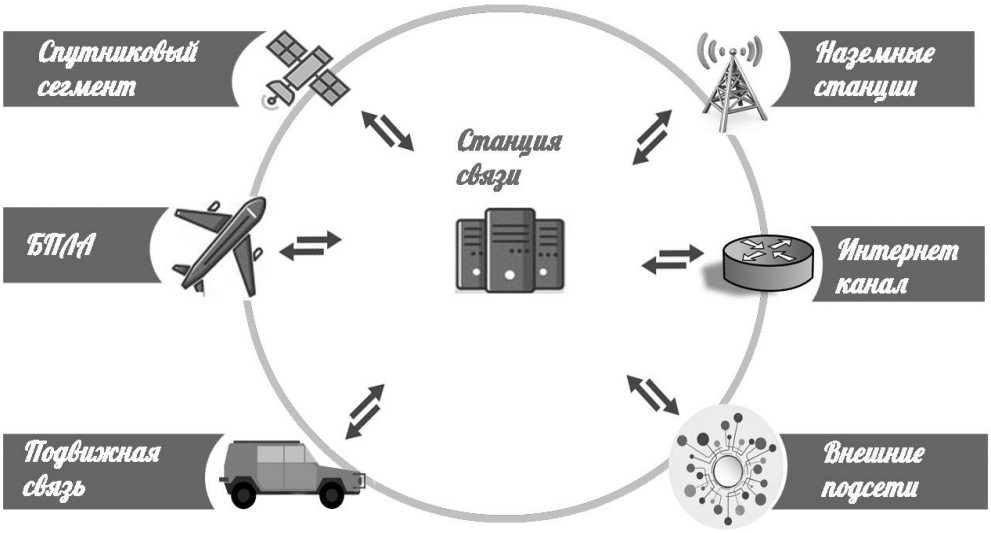



Рисунок 2.8 –Маршрутизируемый комплекс связи

В данном случае признаки были сформированы на основе экспертной оценки параметров системы, которые можно извлечь при маршрутизации. В качестве признаков для определения корректной работы устройства в составе информационной системы, которые будут участвовать в формировании статистики с помощью технологической сети были выбраны следующие параметров [36, 43, 70, 108]:

– $M_a = (m_a^1, m_a^2, ..., m_a^n), n \in \overline{1,N}$, где $M_a$ – вектор адресов памяти, выполняемых на изучаемом устройстве в момент подключения средств диагностики;

– $W_s = (w_a^1, w_a^2, ..., w_s^n), n \in \overline{0,1}$, где $W_s$ – вектор погодных условия для времени проведения сеанса, представляющий собой качественную характеристику (плохая/хорошая);

– $R_{\text{int}} = (r_{\text{int}}^1, r_{\text{int}}^2, ..., r_{\text{int}}^n), n \in \overline{1,k}$, где $R_{\text{int}}$ – вектор выбора типа интерфейса передачи данных во время маршрутизации с помощью алгоритма обмена;

– $C_d = (c_d^1, c_d^2, ..., c_d^n), n \in \overline{1,k}$, где $C_d$ – вектор объема передаваемых данных (выбирается градация – малый, средний или большой набор);

– $N_{tx} = (n_{tx}^1, n_{tx}^2, ..., n_{tx}^n), n \in \overline{1,k}$, где $N_{tx}$ – вектор номеров передающего устройства;

– $N_{rx} = (n_{rx}^1, n_{rx}^2, ..., n_{rx}^n), n \in \overline{1,k}$, где $N_{rx}$ – вектор номеров приемного устройства;

– $L_{time} = (l_{time}^1, l_{time}^2, ..., l_{time}^n), n \in \overline{1,k}$, где $L_{time}$ – вектор меток времени начала сеанса обмена между устройствами;

– $Y_{res} = (y_{res}^1, y_{res}^2, ..., y_{res}^n), n \in \overline{0,1}$, где $Y_{res}$ – вектор результатов проведения сеанса.

Исследование данных для информационного комплекса показывает, что признаки возник делятся примерно на 5-7 крупных групп, собранных по отличительным признакам. Приведем таблицу передач (табл. 2.2) между устройствами информационной системы, моделирование которой выполнено для заданного набора векторов данных.

Таблица 2.4 – Сбор статистики в информационном комплексе при моделировании

| | **Моделирование обмена** Modelling feature | | | | | | |
|---|---|---|---|---|---|---|---|
| *Результат* | *Передача* | *Прием* | *Объем данных* | *№ Инт.* | *Погода* | *Адрес памяти* | *Дата и время* |
| 0 | 9 | 7 | Средний | 9 | Хорошая | 135121239 | 2019-02-02 5:37:50 |
| 1 | 6 | 5 | Малый | 3 | Хорошая | 134258225 | 2018-04-03 22:19:37 |
| 1 | 8 | 6 | Малый | 7 | Плохая | 135153655 | 2019-07-22 13:46:23 |
| 0 | 7 | 5 | Средний | 7 | Хорошая | 134589502 | 2018-10-16 3:43:39 |
| 1 | 2 | 8 | Средний | 5 | Плохая | 135071493 | 2019-11-10 7:02:08 |
| 1 | 7 | 1 | Малый | 6 | Хорошая | 134277113 | 2018-09-22 9:06:26 |
| 1 | 2 | 8 | Малый | 8 | Плохая | 134534935 | 2019-09-29 9:54:51 |
| 1 | 5 | 5 | Средний | 2 | Хорошая | 135000506 | 2018-10-31 17:24:29 |
| 1 | 9 | 3 | Малый | 4 | Хорошая | 134890197 | 2019-07-30 3:31:36 |
| 0 | 3 | 3 | Малый | 1 | Хорошая | 134332480 | 2019-06-05 22:45:28 |
| … | … | … | … | … | … | … | … |
| 0 | 5 | 6 | Большой | 1 | Плохая | 135162910 | 2018-11-17 1:02:20 |
| 0 | 7 | 6 | Средний | 4 | Плохая | 134646037 | 2019-04-29 20:38:15 |
| 1 | 5 | 1 | Средний | 1 | Хорошая | 134382516 | 2019-08-25 4:18:39 |

Пусть имеется набор узлов, которые требуется соединить кратчайшим по времени путем, исключая из рассмотрения отказавшие узлы и учитывая оперативно добавленные интерфейсы. Установим следующие обозначения: $A=\{a_j\}$ – множество узлов в комплексе связи; $a_j$, $j=\overline{1,N}$ – узел; N=|A| – количество узлов в комплексе; $I=\{i_m\}$ – множество интерфейсов узла; $i_m$, $m=\overline{1,M}$ – интерфейсы узла; N=|*I*| – количество интерфейсов узла; $U_{en}=\{a_j : d_j \geq d_{\min}\}$ – множество доступных узлов; $d_{min}$ – признак стабильной работы интерфейса; $U_n=\{a_j : d_j < d_{\min}\}$ – множество недоступных узлов. Требуется минимизировать время передачи сообщения от узла-инициатора до узла-приемника:

$$T=\sum t_{kj}; t_{kj}=t_{kj}^{pr}+t_{kj}^{tx}; k=\overline{1,N-1}, k\neq j, \tag{2.9}$$

где $t_{kj}^{pr}$ – время обработки в *j*-м узле, доступном для коммуникации с *k*-м узлом; $t_{kj}^{tx}$ – время передачи между узлами *k* и *j*; $a_j \in U_{en}$.

Функция поиска узла-приемника позволяет перейти к следующему промежуточному узлу на графе:

$$F_{next}=j, \{j : t_{kj}(j)=\min t_{kj}\}. \tag{2.10}$$

Функция достижения результата осуществляет проверку на достижения узла-приемника $a_n$

$$F_{end}(a_n)=1; F_{end}(a_n)=0,\ j\neq n. \tag{2.11}$$

Алгоритм предназначен для использования на узле передачи и обеспечивает простую логику для выбора маршрута до конечного узла. Так как существует множество методов поиска кратчайшего пути, требуется выбрать тот, который позволит быстрее достичь заданной цели. Для обоснованного выбора алгоритма требуется аппарат сравнения. В теории сложности вычислений вычислительная сложность любого компьютерного алгоритма оценивается с помощью четырёх основных показателей: полнота, временная сложность, пространственная сложность, оптимальность. В нашем случае для решения поставленной задачи достаточно оценить алгоритмы по временной сложности (оценка времени работы

алгоритма). Классические алгоритмы нахождения оптимального пути в общем случае имеют временную сложность, описанную квадратичным уравнением относительно входного количества вершин графа для обработки. Более поздний эвристический алгоритм поиска на графе по наилучшему совпадению A star обладает временной сложностью, описанной логарифмическим уравнением, но не при всех значениях количества входных вершин. Существуют состояния, где он описывается экспоненциальной зависимостью [30].

Алгоритм JPS (Jump Pointer Search), разработанный в австралийском центре информационных технологий, является успешной модификацией алгоритма A star, и временная сложность алгоритма описывается логарифмическим уравнением для значительно большего количества вершин. Построение алгоритма на основе JPS позволит сократить издержки на время поиска пути до конечного узла. Так как целевое состояние системы зависит не только от географического пути, но и от интерфейса, то целевая функция должна учитывать это обстоятельство. В связи с вышесказанным, ставится дополнительная задача определить самые быстрые интерфейсы узла. Под каналом связи будем понимать канал обмена между двумя соседними узлами по выбранному интерфейсу. Специфика выделения основного интерфейса узла в комплексе связи состоит в том, что канал связи является динамическим, зависит от погодных условий, поддерживаемого протокола, наличия у соседнего узла такого же интерфейса (иначе связь по интерфейсу невозможна), поэтому проводится ранжирование интерфейсов между собой. Целевая функция алгоритма должна выбирать оптимальный путь – набор интерфейсов и узлов с минимальным временем. На канал связи влияют группы параметров, выделение которых происходит экспериментально. Определение зависимости скорости связи от групп параметров канала выполняется на основе вычисления условной энтропии. Применительно к работе комплекса связи под событием понимается появление фактора, влияющего на связь (помехи в канале связи, отказ интерфейса, внешние воздействия и др.). Чтобы сравнивать скорости различных интерфейсов требуется получить шкалу оценки. Минимальным значением для шкалы является минимальная скорость интерфейса с меньшей

скоростью и частыми сбоями, а максимальной – скорость самого производительного интерфейса в комплексе с лучшими параметрами канала. После оценки имеющихся интерфейсов комплекса требуется добавить приоритет выбора интерфейса по скорости как ограничение целевой функции на этапе выполнения алгоритма. Учитывая возможность обрыва или неполадки интерфейса, перезапустить процесс маршрутизации способен каждый узел, выбранный в качестве целевого, т.е. при обрыве связи промежуточный узел запускает задачу поиска кратчайшего пути, где уже он является отправителем. Алгоритм JPS используется узлом для расчета целевой функции при учете поставленных ограничений.

Также следует отметить, что количество узлов связи для полученного алгоритма на порядок ниже, откуда следует, что выбранные интерфейсы наиболее предпочтительны для передачи информации. Такое знание полезно для последующей модификации алгоритма. В дальнейшем, можно пересылать между соседними узлами информацию о выгодных интерфейсах во время отсутствия трафика. Под точками отсечения понимаются точки между начальной и конечной, переходы между которыми невозможны. Алгоритм состоит из 7 этапов:

На *первом этапе* формируются таблицы переходов между объектами маршрутизируемой сети с помощью функции поиска приемника для набора узлов между начальным и конечным. Для этого вычисляется энтропия для событий, влияющих на связь по формуле:

$$G(X) = -\sum_{i=1}^{N} p_i \sum_{j-1}^{M} p_j \log_2 p_i(j)\,. \tag{2.12}$$

Также рассчитывается информация о разбросе значений скорости и ее отклонения:

$$D(U) = \sum_{i=1}^{N} p_i (U_i - (\sum_{i=1}^{N} p_i U_i))^2, \tag{2.13}$$

где $U$ – среднее значение скорости, $D$ – функция разброса $U$, $N$ – количество накопленных значений скорости, $U_i$ – $i$-ое значение скорости узла, $p_i$ – вероятность того, что скорость примет значение $U_i$.

На основании полученных характеристик проводится ранжирование точек по минимальной стоимости функции достижения относительно начальной точки. На *втором этапе* для работы алгоритма формируется таблица узлов в виде квадратной матрицы. При меньшем количестве переходов, чем размер матрицы, отсутствующие узлы добавляются в множество точек отсечения.

На *третьем этапе* для заданного промежутка времени проводят вычисление целевой функции скорости для алгоритма MJPS по формуле и полученные значения сохраняют в базу данных:

$$D(\Delta U) = \frac{1}{L}\sum_{i=1}^{L}(U_i - \frac{1}{L}\sum_{i=1}^{L}U_i)^2, \tag{2.14}$$

где $\Delta U$ – отклонение от среднего значения скорости, $D$ – функция разброса $\Delta U$, $U_i$ – $i$-ое значение скорости узла, $L$ – число измерений скорости выбранного узла.

На *четвертом этапе* моделируются дополнительные точки, соответствующие альтернативным интерфейсам для узлов.

На *пятом этапе* производят определение кратчайшего пути по алгоритму MJPS со значениями рассчитанных скоростей и результаты заносятся в базу данных.

На *шестом этапе* выполняется переход от одного узла к другому, для которого проверяется появление точек отсечения и при необходимости инициируется новый расчет пути.

На *седьмом этапе* проверяется достижение конечного узла и при невыполнении этого условия сверяется информация об отказе промежуточных интерфейсов связи, формируется набор точек отсечения для возврата на 2-й этап. При достижении узла происходит алгоритма заканчивает работу. На рисунке 2.9 и 2.10 приведены варианты работы алгоритма маршрутизации без отсечений, а на рисунке 3 с отсечением точек, где (а) – алгоритм Дейкстры, (б) – алгоритм A star, (в) – алгоритм MJPS. N – количество операции, t – время выполнения в миллисекундах [45].

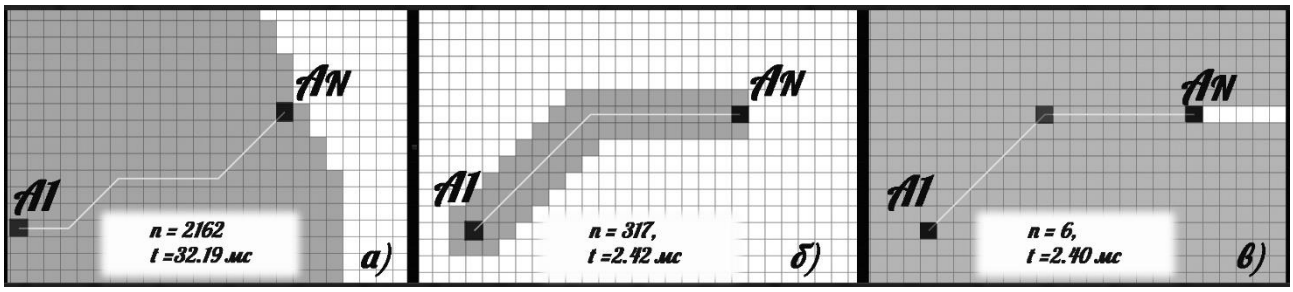



Рисунок 2.9 – Сравнение количества операций *n* и времени поиска пути *t* для заданных точек при использовании без отсечений

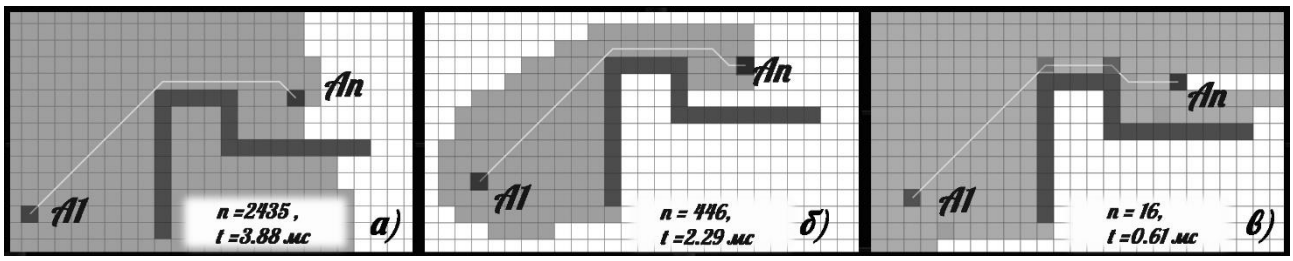



Рисунок 2.10 – Сравнение количества операций *n* и времени поиска пути *t* для заданных точек с отсечениями

Видно, что предлагаемый алгоритм рассматривает меньшее количество точек (выделены на линии движения цветом) по сравнению с двумя другими алгоритмами. Кроме того, для алгоритмов указаны требуемые для решения задачи количество операций *n* и затрат по времени на поиск пути *t* для заданных точек $A_1$ – $A_N$. Так как предлагаемый алгоритм выполняется за меньшее время, чем приведенные алгоритмы при сравнимых количествах операций, то его использование предпочтительнее. Информационная система состоит из устройств, которые обладают набором разнородных интерфейсов. При применении технологической сети формируется статистика работы комплекса в условиях приема-передачи информации между узлами. Технологическая сеть позволяет выполнить экспертную диагностику работы устройств, на основе признаков установить корректность работы. Сбор статистики осуществляется во время работы системы и связан с алгоритмами работы микропроцессорного устройства, который накапливает набор решений по адаптивной к условиям среде информации.

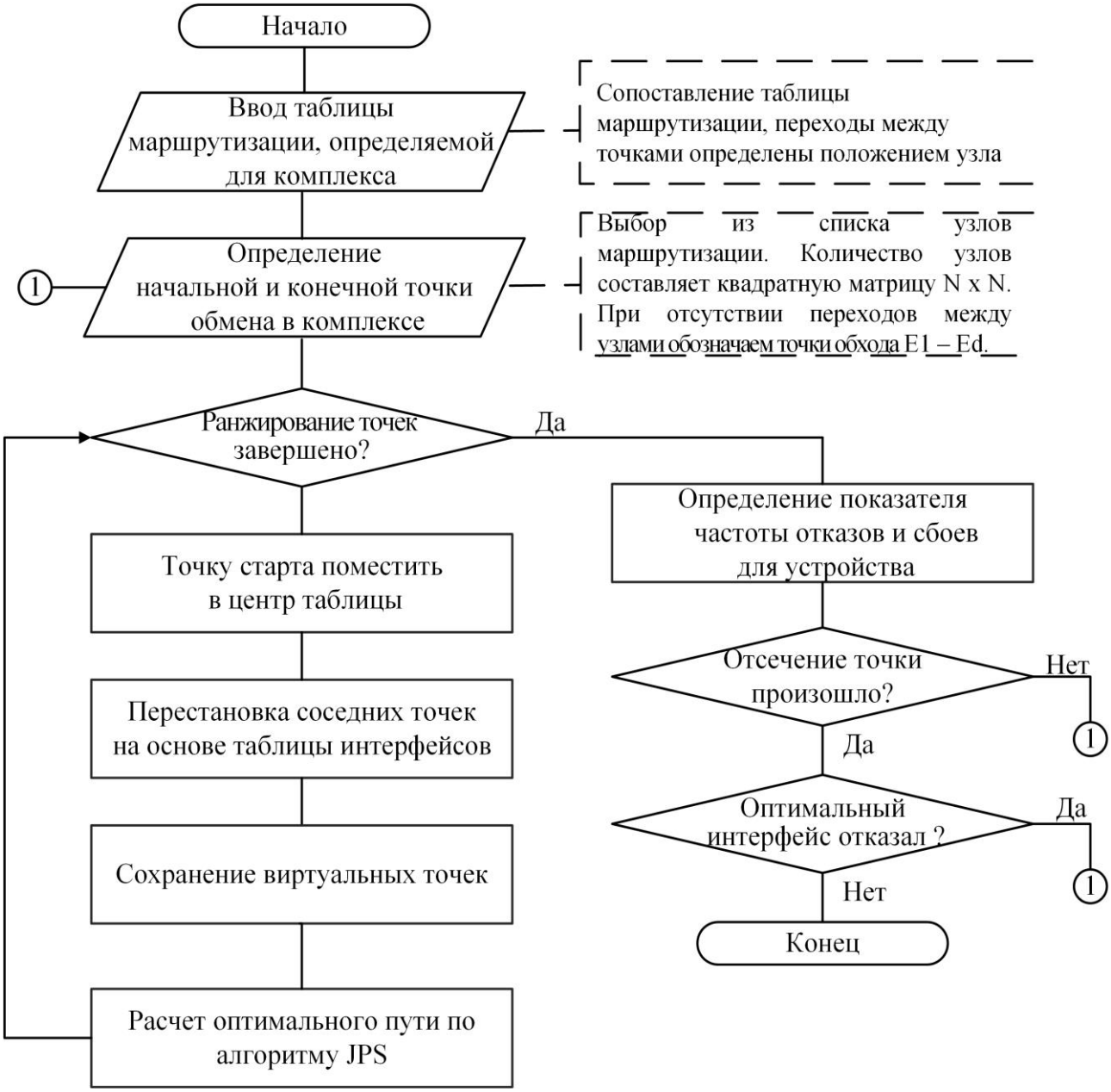
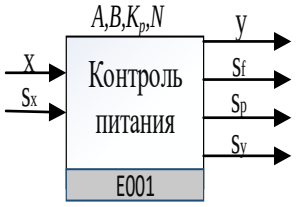
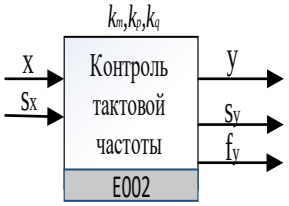
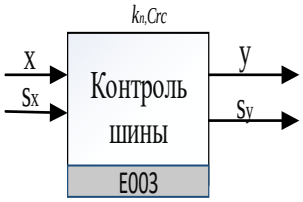


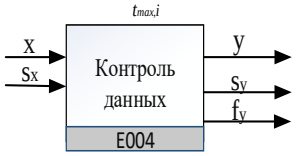
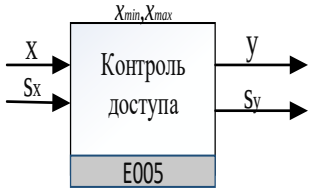
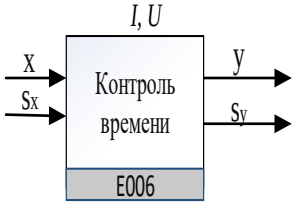

Рисунок 2.11 – Алгоритм адаптивной передачи информации

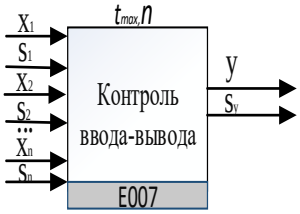
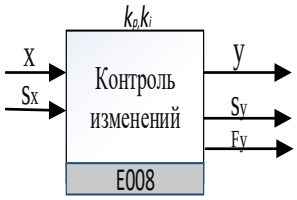
## 2.4 Разработка модуля обнаружения аппаратных неисправностей в составе программного блока имитации отказов и сбоев

Работа модулей выполняется вместе с тестовыми программами, изменяющими значения параметров, которые контролирует модуль. В таблице 2.5 приведены основные модули, которые могут быть использованы для выявления возникающих отказов и сбоев системы [35, 42, 57].

Таблица 2.5 – Модули контроля системы

| Наименование модуля<br>Изображение на схеме | Входы и выходы модуля<br>Параметры настройки | Алгоритм работы |
|---|---|---|
| 1. Модуль контроля питания<br>  <br>***Параметры настройки:***<br>$k_p$ – коэффициент напряжения питания;<br>$k_n$ – размер буфера для замеров напряжения питания;<br>$A$ – нижняя граница значения питания;<br>$B$ – верхняя граница значения питания | ***Входы:*** **x** – величина напряжения питания;<br>$S_x$ – признак недостоверности входного сигнала.<br>***Выходы:*** $y$ – функция контроля питания;<br>$S_f$ – признак отклонения среднего значения питания от граничных значений;<br>$S_p$– признак отклонения величины напряжения от граничных значений;<br>$S_y$ – признак достоверности функции контроля | $y = \sum_{i=1} k_p x_{t+i} \cdot k_n^{-1}$<br>$S_f = \begin{cases} 0 \text{ при } y \le (B-A) \\ 1 \text{ при невыполнении} \end{cases}$<br>$S_p = \begin{cases} 0 \text{ при } A \le x_t \le B) \\ 1 \text{ при невыполнении} \end{cases}$<br>$S_y = \begin{cases} 0 \text{ при } s_f = 0, s_p = 0, s_x = 0 \\ 1 \text{ при невыполнении} \end{cases}$ |
| 2. Модуль контроля тактовой частоты шин<br>  <br>***Параметры настройки:***<br>$k_m$ – значение частоты шины 1<br>$k_p$ – значение частоты шины 2<br>$k_q$ – значение частоты шины 3 | ***Входы:*** $x$ – значение тактовой частоты на входе микроконтроллера;<br>$S_x$ – признак достоверности входной частоты.<br>***Выходы:*** $y$ – реальное значение тактовой частоты;<br>$S_y$ – признак достоверности выходной частоты;<br>$f_y$ – функция сравнения основного и дублирующего выхода | $y = \begin{cases} k_m\ k_p k_q^{-1} x \text{ при } S_x = 0 \\ 1 \text{ при невыполнении} \end{cases}$<br>$S_y = \begin{cases} 0 \text{ при } S_x = 0,\ f_y = 0 \\ 1 \text{ при невыполнении} \end{cases}$<br>$f_y = \begin{cases} 0 \text{ при } k_m k_p k_q^{-1} xy^{-1} < 0.1 \\ 1 \text{ при невыполнении} \end{cases}$ |
| 3. Модуль контроля шины данных<br>  <br>***Параметры настройки:***<br>$k_n$ – заданное число тактов шины; $Crc$ –функция вычисления контрольной суммы | ***Входы:*** $x$ – значение сигнала на шине данных;<br>$S_x$ – признак достоверности установленной частоты.<br>***Выходы:*** $y$ – полученные данные за $k_n$ тактов;<br>$S_y$ – признак недостоверности данных | $S_y = \begin{cases} 0 \text{ при } y = Crc(x(k_n - 4)) \text{ и } S_x = 0 \\ 1 \text{ при невыполнении} \end{cases}$ |

| Наименование модуля<br>Изображение на схеме | Входы и выходы модуля<br>Параметры настройки | Алгоритм работы |
|---|---|---|
| 4. Модуль контроля корректности данных<br>  <br>***Параметры настройки:***<br>$t_{max}$ – максимальное время; реакции контроллера на заданной частоте;<br>$i$ – число проверок модуля | ***Входы:*** $x$ – значение сигнала на входе контроля;<br>$S_x$ – признак потери или повреждения данных.<br>***Выходы:*** $y$ – функция сравнения обработанных данных с входным эталоном $x$;<br>$Sy$ – признак отказа контроллера во время операции приема данных.<br>$f_y$ – функция сравнения входного и выходного сигнала | $y_t = \begin{cases} x_t \text{ при } S_x = 0 \\ y_{t-t_{\max}} \text{ при невыполнении} \end{cases}$<br>$f_y = \begin{cases} 0 \text{ при } \sum_i y_i^{-1} x_i > \frac{1}{2} \\ 1 \text{ при невыполнении} \end{cases}$<br>$S_y = \begin{cases} 0 \text{ при } S_x = 0 \text{ и } f_y = 0 \\ 1 \text{ при невыполнении} \end{cases}$ |
| 5. Модуль нарушения прав доступа к памяти.<br>  <br>***Параметры настройки:*** $x_{min}$ – младший доступный адрес памяти;<br>$x_{max}$ – старший доступный адрес памяти. | ***Входы:*** $x$ – адрес обращения к области памяти;<br>$S_x$ – признак привилегированного доступа для x.<br>***Выходы:*** $y$ – функция проверки корректной обработки доступа;<br>$S_f$ – признак отклонения среднего значения от граничных значений | $y = \begin{cases} 0 \text{ при } x_{\min} \le x \le x_{\max} \\ 1 \text{ при невыполнении} \end{cases}$<br>$S_y = \begin{cases} 0 \text{ при } S_x = 0 \text{ и } y = 0 \\ 1 \text{ при невыполнении} \end{cases}$ |
| 6. Модуль анализа времени выполнения<br>  <br>***Параметры настройки:***<br>$I$ – номер интерфейса передачи;<br>$U$ – регулируемая скорость передачи | ***Входы:*** $x$ – входной сигнал данных по интерфейсу $I$;<br>$S_x$ – признак отказа интерфейса передачи данных.<br>***Выходы:*** $y$ – время передачи байта данных;<br>$S_y$ – признак отклонения времени от допустимых значений | $y_t = \begin{cases} t \text{ при } (x_t - x_{t-1})U^{-1} \le t_{\max} \\ y_{t-t_{\max}} \text{ при невыполнении} \end{cases}$<br>$S_y = \begin{cases} 0 \text{ при } y = 0 \text{ и } S_x = 0 \\ 1 \text{ при невыполнении} \end{cases}$ |

| Наименование модуля<br>Изображение на схеме | Входы и выходы модуля<br>Параметры настройки | Алгоритм работы |
|---|---|---|
| 7.Модуль контроля ввода-вывода<br>  <br><br>***Параметры настройки:***<br>$t_{max}$ – максимальное время реакции порта ввода-вывода.<br>$n$ – число проверяемых портов ввода-вывода | ***Входы:*** $x$ – входной сигнал;<br>$S_1$ – $S_n$ – признак запрещения доступа к портам ввода-вывода.<br>***Выходы:*** $y$ – изменение выходного сигнала по входному сигналу;<br>$S_y$ – признак отказа порта ввода-вывода | $s_y = \begin{cases} 0 \text{ при } y_{t+t_{max}} = (X_t \bmod n) \text{ и } S_1,\ldots,S_n = 0 \\ 1 \text{ при невыполнении} \end{cases}$ |
| 8. Модуль контроля вносимых изменений<br>  <br><br>***Параметры настройки:***<br>$k_p$ – признак внесения изменения в память процессора;<br>$k_i$ – признак внесения изменения в интерфейс | ***Входы:*** $x$ – сигнал вносимого изменения;<br>$S_x$ – признак отказа доступа по адресу.<br>***Выходы:*** $y$ – функция детектирования изменений;<br>$S_y$– признак корректности реакции;<br>$f_y$ – признак отказа интерфейса | $y_t = \begin{cases} 0 \text{ при } S_x = 0,\ f_y = 0 \\ 1 \text{ при } S_x = 1 \text{ и } f_y = 1 \\ 2 \text{ при } S_x = 1 \text{ и } f_y = 2 \end{cases}$<br>$f_y = \begin{cases} 0 \text{ при невыполнении} \\ 1 \text{ при } x_t = k_i \\ 2 \text{ при } x_t = k_p \end{cases}$<br>$S_y = \begin{cases} 0 \text{ при } f_y = 1, y_t = 0;\ f_y = 2, y_t = 0 \\ 1 \text{ при невыполнении} \end{cases}$ |

Модули контроля системы позволяют зафиксировать отказы и сбои во время работы тестовой инфраструктуры, но не во всех случаях. Для последнего тестируемого модуля не удалось детектировать отказы, поэтому для эффективности обнаружения отказов и сбоев планируется использование модулей обнаружения совместно.

## 2.5 Разработка модуля выявления ошибок программного обеспечения в составе программного блока имитации отказов и сбоев

Тестируемая система представлена набором микроконтроллеров (или, как вариант, персональным компьютером). Управляющая система состоит из пяти компонент. Главный компонент управляющей станции – рабочая станция (PC)

имитации неисправностей, которая производит процедуры запуска, коррекции и остановки процесса имитации неисправностей. Рабочая станция баз данных (РС БД) предназначена для сохранения результатов (статистики) проведения испытаний. Контроллеры диагностики и устройство внешнего контроля представляют собой набор специальных средств с функциями чтения и записи в память исполняемых программ по интерфейсу *JTAG*.

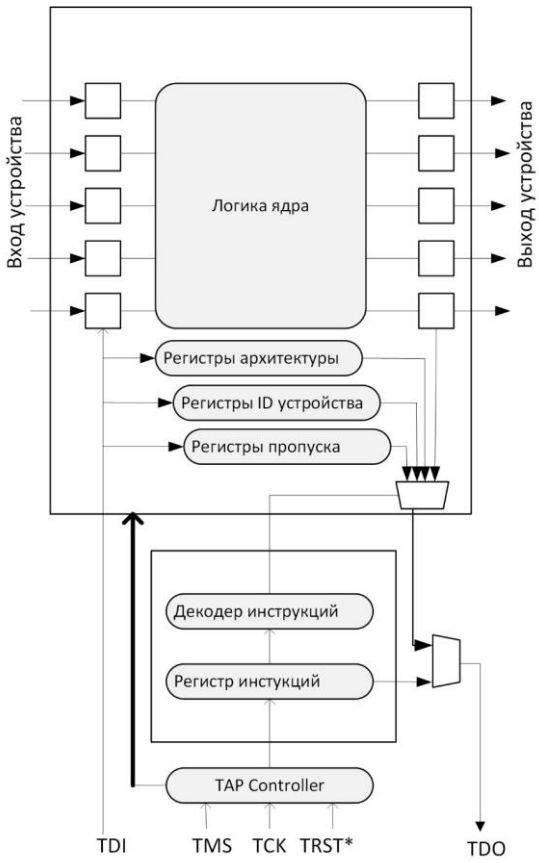



Рисунок 2.12 – Контроллер диагностики на основе JTAG

Данный интерфейс позволяет осуществлять подключение к современным типам микросхем и осуществлять протокол внутрисхемной отладки к доступам к памяти и регистрам. На рисунке приведена типовая схема организации JTAG-интерфейса [89]. JTAG-интерфейс содержит через четыре обязательных контакта (TDI, TDO, TCK, TMS) и один дополнительный контакт (TRST): TDI (Test Data In): ввод данных; TDO (Test Data Out): вывод данных; TCK (Test Clock): часы, максимальная частота которых зависит от микросхемы; TMS (выбор тестового

режима): контакт для управления конечным автоматом; TRST (Test Reset): дополнительный вывод для сброса конечного автомата.

Интерфейс TAP реализует конечный автомат (16 состояний), который позволяет получить доступ к группе регистров (*IR*, *DR*) для измерения микросхемы. Управление этим конечным автоматом осуществляется через выводы *TMS* и *TCK*. С помощью этого конечного автомата можно выбрать операцию через регистр *IR* (регистр команд) и передать параметры или проверить результат через регистр *DR* (регистр данных).

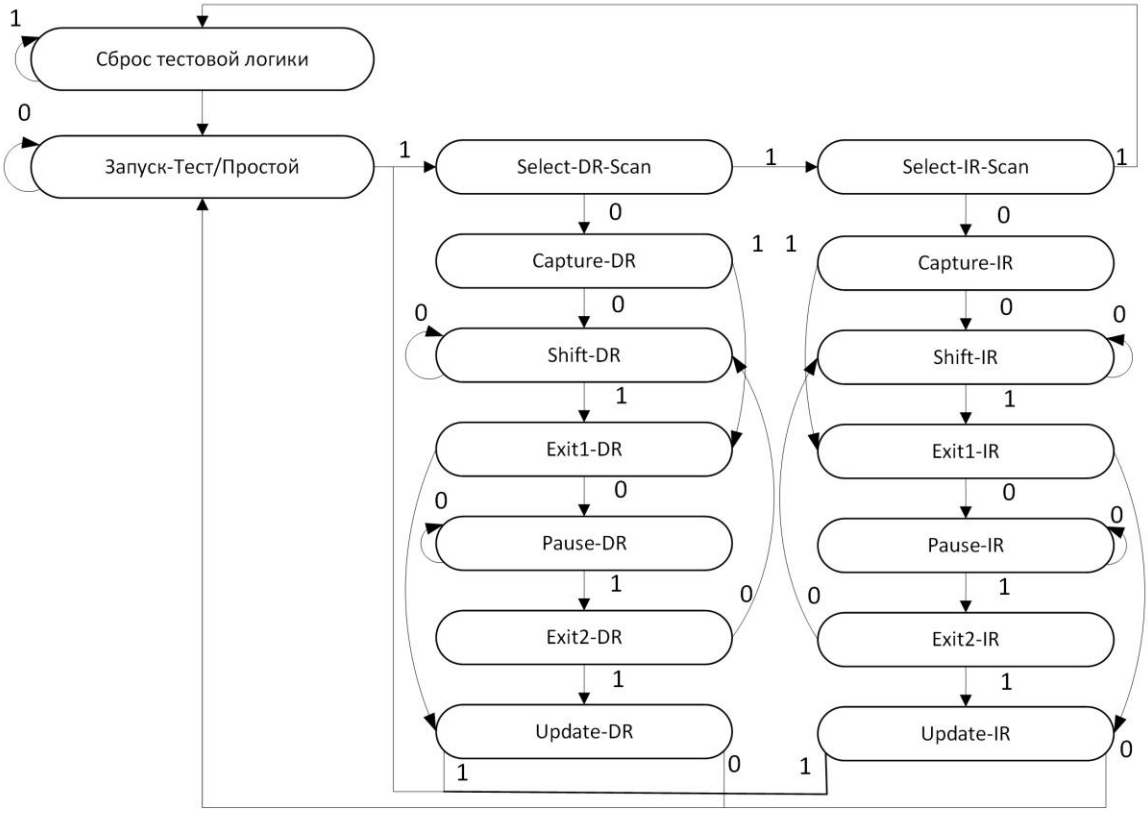



Рисунок 2.13 – Автомат состояний JTAG-контроллера

PC генерирует наборы данных, которые записываются контроллером диагностики в память данных набора микроконтроллеров во время исполнения работы (микроконтроллер с запущенной программой выполняет заданное количество функций). Блоки контроля и диагностики фиксируют реакцию набора микроконтроллеров после внесения изменений и сохраняют информацию в базу данных, после чего PC анализирует сохраненные данные, корректирует по данным процесс имитации. Испытания останавливаются оператором на PC. Программное обеспечение, которое установлено на PC и управляет процессом имитации

неисправностей, рассмотрено ниже. При замене микроконтроллерной системы на персональный компьютер другие блоки выполнены программно, не включая средства диагностики. Устройство обеспечивает автоматизацию и ускорение проведения испытаний за счет сбора и обработки информации о программных функциях, которые выполняются в тестируемой системе.

С помощью контроллера диагностики (по интерфейсу JTAG) собирается статистика о времени, выделяемом микроконтроллером на исполнение конкретных функций в программе. Таким образом, испытания по внесению ошибок в данные производятся сначала в память, обращение к которой происходит в программных функциях, занимающих больше микропроцессорного времени. При имитации неисправностей РС имитации с помощью средств диагностики собирает данные и определяет вероятность выполнения программной функции во время отказа (сбоя), что позволяет при повторной работе имитатора по имеющейся в базе данных информации определить, какая неисправность обнаружена при имитации.

Отдельно следует рассмотреть техническое средство имитации неисправностей, которое представляет собой программно-аппаратный блок устройства имитации. В качестве такого блока используется аппаратные устройства с поддержкой JTAG-интерфейса, которые являются стандартными для большинства микроконтроллеров архитектур. Интерфейс позволяет получить доступ к процессору и интерфейсам заданного микроконтроллера.

Для выполнения мероприятий по тестированию, описанных в стандарте IEC 61508, алгоритмы инъекции могут быть реализованы в рамках представленной системы на станции управления и устройстве имитации. Для проведения имитации неисправностей требуется реализовать внедрение (инъекцию) неисправностей в программно-аппаратную систему. Блок инъектора неисправностей для устройства имеет структуру, представленную на рисунке 2.14. В основном, в блоке используется три гипотезы для формирования тестовых инъекций неисправностей: «White Box» (инъекция в систему с известными характеристиками), «Gray Box» (инъекция в частично изученную систему), «Black Box» (инъекция в неизвестную систему) [114].

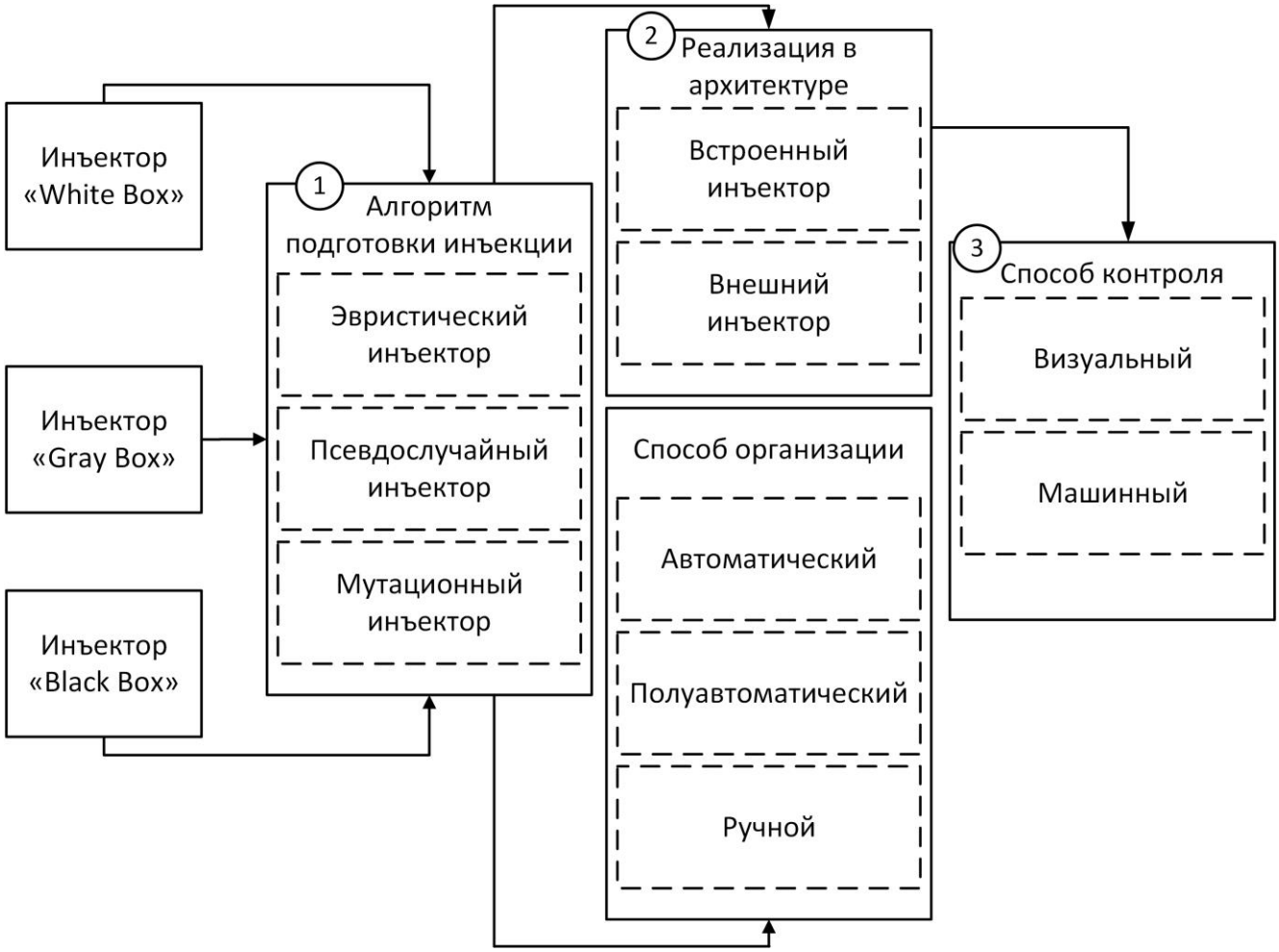



Рисунок 2.14 – Состав устройства имитации (инъектор неисправностей)

Инъектор неисправностей состоит из алгоритма подготовки инъектора, который реализуется на станции управления. Данный алгоритм реализует следующие типы стратегий тестирования [80, 100]:

1.*Эвристический инъектор***.** Тестирование на наличие неисправностей, основанное на знаниях о программно-аппаратных особенностях системы.

2.*Инъектор данных*. Тестирование на наличие неисправностей, основанное на изменении входных данных, которое эксплуатирует проблему робастности.

3.*Мутационный инъектор*. Тестирование на наличие неисправностей, основанное на псевдослучайных изменениях исходного кода и проверке реакции на эти изменения набора автоматических тестов.

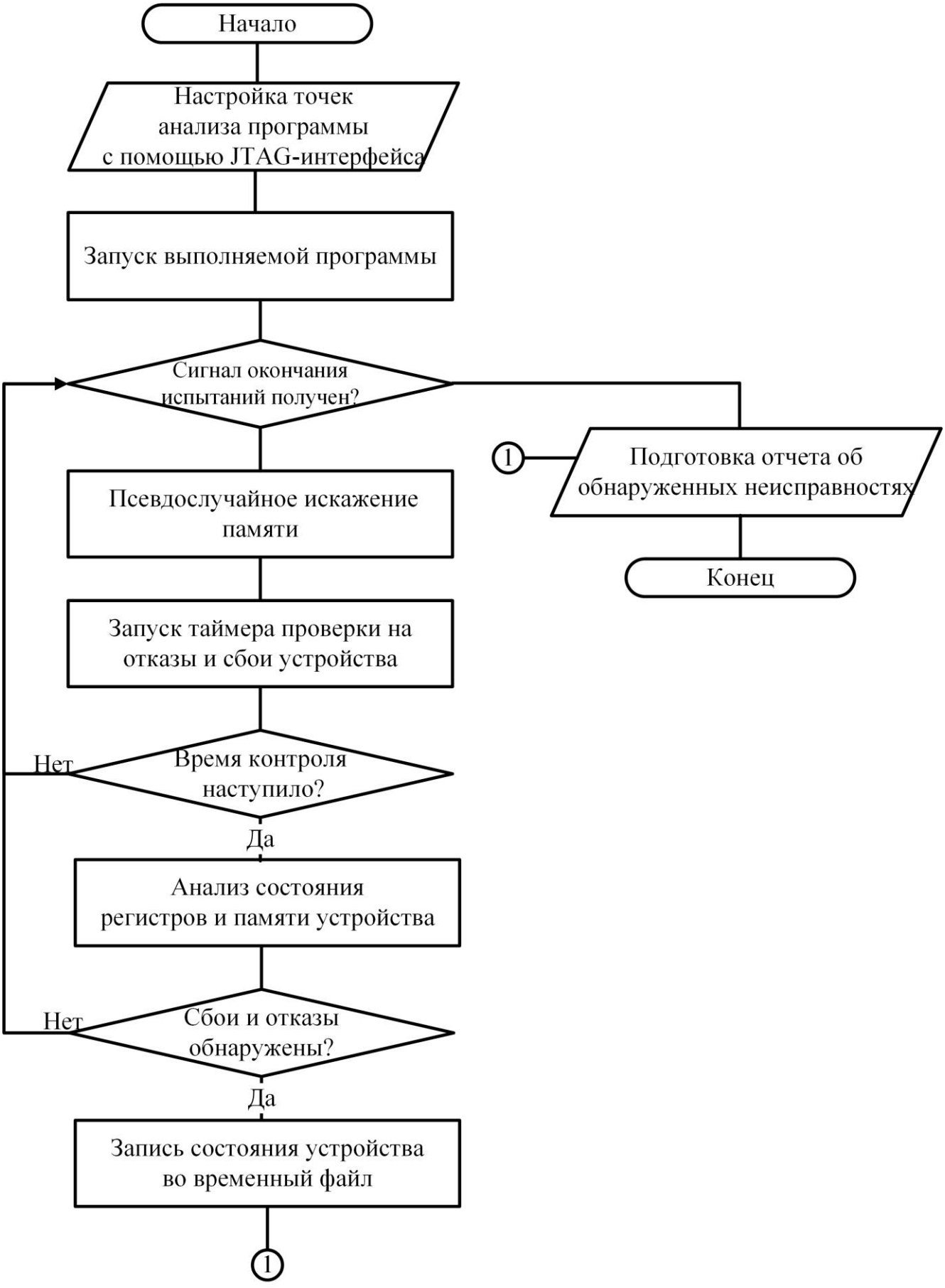



Рисунок 2.15 – Организация тестирования устройства с помощью JTAG

Для целей проводимого исследования приведем события, обнаружение которых позволит более детально изучить работу микропроцессора и классифицировать обнаруживаемые неисправности для построения в дальнейшем для корректировки программного и аппаратного обеспечения. Инъекция имитируемой неисправности имеет цель получить отказ или сбой, фиксируемый с помощью анализа состояния процессора. Инъекция может привести к следующим вариантам последствий для процессора:

– отказ процессора, остановка процессора в данном состоянии и полная неспособность устройства работать без длительных действий по восстановлению;

– отказ процессора, остановка процессора в данном состоянии и полная неспособность устройства работать без кратковременной перезагрузки;

– периодический неконтролируемый сбой процессора;

– периодический сбой процессора при выполнении функции;

– однократный определяемый сбой процессора;

– отсутствие последствий внесения изменений для устройства.

Анализ реакции на имитируемые неисправности производится с помощью средств к модификации которых есть доступ у разработчика – регистры процессора и интерфейсов, данные, хранимые в ОЗУ, и шаблоны выполняемых функций, сохранённые до проведения инъекции.

## 2.6 Выводы по главе

1. Рассмотрены методы имитации неисправностей для цифровых систем. Показано, что использование режимов искажения входных данных является эффективным инструментом для тестирования программного обеспечения в целевой аппаратной среде. Предложена структура программно-аппаратного устройства для имитации неисправностей в проектируемых системах на базе микропроцессорных средств, позволяющего производить инъекции ошибок в уязвимые места ПО: входные данные и регистры памяти, задействованные при выполнении основных программных функций.

2. Исследованы методы автоматизации искажения данных и анализа реакции на внесенные ошибки в проектируемую систему. Рассмотрены различия между результатами тестирования на основе (1) внесения неисправностей в функциональные компоненты устройства (блоки памяти, аппаратные интерфейсы, элементы системного взаимодействия) и на основе (2) имитации последствий возникновения неисправностей. Показано, что использование второго метода (для которого реализован программный модуль) имеет преимущества для микропроцессорных систем в связи с отсутствием разрушающих действий и уменьшением временных затрат на его осуществление.

3. Предложена структура и описаны функции для аппаратного блока обнаружения аппаратных неисправностей в составе программного блока имитации отказов и сбоев, позволяющего классифицировать дефекты аппаратных средств.

4. Предложен алгоритм модуля обнаружения отказов и сбоев для диагностики программно-аппаратного комплекса, который позволяет оптимизировать скорость передачи данных путем поиска кратчайшего пути в совокупности с ранжированием интерфейсов приема/передачи при наличии большого числа интерфейсов (узлов).

# 3 РАЗРАБОТКА СТРУКТУРЫ И АЛГОРИТМОВ КОМПЛЕКСА ПРОВЕДЕНИЯ ИСПЫТАНИЙ МИКРОПРОЦЕССОРНЫХ УСТРОЙСТВ

Третья глава **представляет** разработанные алгоритмы проведения испытаний микропроцессорного устройства, которые позволяют получить критерий окончания тестирования для обеспечения работоспособности проектируемого устройства, а также программные средства фаззинга для проведения исследований.

## 3.1 Структура комплекса проведения испытаний микропроцессорных устройств

Рассмотрим особенности существующих аналогов в области имитации неисправностей. Известен способ [46], включающий мутационное тестирование радиоэлектронной аппаратуры и ее управляющего программного обеспечения, который использует оценку вероятности обнаружения имитируемых моделей неисправностей тестируемой аппаратуры или программы с помощью имитации поведения каналов ввода-вывода объекта испытаний. Для микропроцессорного устройства данный вариант технически не применим в связи со сложностью использования для микроконтроллерных ядер, которые являются собственностью компаний-производителей и не могут быть смоделированы на ПЛИС. К тому же у способа тестирования отсутствует четкая методика окончания испытаний, которая зависит от модели неисправностей и не позволяет гарантировать конечность проведения испытаний.

При этом типичный состав устройства информационного комплекса (микропроцессорного устройства) включает интерфейс JTAG, позволяющий получать в режиме отладки точную информацию о состоянии памяти, регистров, стека и времени выполнения команд на устройстве. При соединении с испытаниями электронной схемы на отказоустойчивость представляет осуществить проверку с помощью внешних устройств контроля проектируемой инфраструктуры тестирования, заключающуюся в имитации отказов элементов

при сохранении работоспособности схемы, что позволяет в автоматическом режиме фиксировать отказы схемы по выходным сигналам. В данной работе предлагается расширить данные способы и применить на проектируемом имитаторе неисправностей для микроконтроллера как составной части проектируемого устройства.

Такая система с программным обеспечением, имеющим режимы работы по прерыванию и DMA-режим, позволит сравнивать сложные режимы обработки на выходе одновременно нескольких результатов, обеспечивающих правильный результат. Критерием отказа системы может быть целый набор косвенных результатов выполнения функций устройства. Для проектируемой методики необходимо осуществить использование модели испытаний, основанных как на знании внутренней структуры устройства (модель «белого» или «серого» ящика) и реализации механизмов имитации, так и на основе «черного» ящика, что позволит исследовать составные микросхемы, внутреннее устройство которых не неизвестно (современные системы включают большое количество интегральных микросхем с неизвестной внутренней структурой, не раскрываемой фирмами-производителями). Использование модели «черного» или «серого» ящика при неизвестной модели СБИС позволяет испытывать системы с неизвестной внутренней структурой, которая не может быть получена от производителя.

В данной главе будет представлена методика, задачей которой является определение объема испытаний по имитации неисправностей, необходимого для выявления отказов и сбоев с заданной точностью. Результат применения такой методики будет состоять в автоматизации испытаний и сокращении времени испытаний, имитации последствий отказов и сбоев элементов, уменьшении вероятности отказов системы за счет выявления элементов, отказ которых приводит к неработоспособности системы, определении объема испытаний, необходимого для выявления отказов и сбоев с заданной точностью и на базе спроектированного устройства для имитации неисправностей в программно-аппаратных системах.

Для и скажения данных программы используется три основных подхода. Первый заключается в генерационной схеме применения фаззинга, т.е. существует специальный модуль контроля и настройки программы фаззинга, который на основе статистики формирует данные, обрабатываемыми программой. Фаззинг [22, 66, 69, 85, 88, 94] является специальным методом для тестирования сложных систем, который проверяет устойчивость работы алгоритма к различным наборам входных данных. Устойчивость системы к неопределенным данным (свойство робастности программного обеспечения) исследуется в условиях эксплуатации программы для алгоритмов работы устройства.

Используем следующее обозначения: $b_x$ – признак возникновения ошибки в программе для данных *X;* $t_{thres}$ – время на осуществление фаззинга*;* $c_{ij}$ – уникальный *i*-й сбой или отказ для *j*-го набора данных*; f* – уникальная программная функция, $\mathbb{C}$ – набор входных данных для мутаций. Функция *Fuzz* выполняет искажение данных по определенным сценариям [119]

$$Fuzz(\mathbb{C}, t_{thres}) = \{(b_1, S_1, t_1), \ldots, (b_n, S_n, t_n)\}. \quad (3.1)$$

Различные комбинации входных данных позволяют получить уникальный граф исполнения программы (вершинами которого являются программные функции графа, а ребра графа соединяют вершины в порядке вызова функций), для которого выполняется проверка работы [37, 104, 106].

Целью проведения фаззинга является получение максимального числа ошибок за установленный временной лимит, а также максимизация обхода ветвей графа программы. Как показано в работах других авторов, фаззинг позволяет решать задачу об оптимизации количества найденных уязвимостей [95, 96, 114] (покрытия программы) за установленный временной диапазон.

$$\sum_{i=1}^{N} b_x \to \max . \quad (3.2)$$

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

$$c_{ij} = \begin{cases} 1, \ j\text{-й } \textit{отказ включает } i\text{-й } \textit{набор данных}; \\ 0, \ \textit{в противном случае.} \end{cases} \quad (3.3)$$

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

$$b_x = \begin{cases} 1, \textit{тогда и только тогда, когда } \exists i, j : f(c_{ij}) = x; \\ 0, \textit{в противном случае.} \end{cases} \quad (3.4)$$

При этом следует учесть ряд ограничений для достижения максимального числа ошибок за установленное время поиска:

$$\begin{gathered} \forall i, j : c_{i,j+1} \le c_{i,j}; \\ \sum_{i.j} c_{ij} \cdot t_{ij} \le t_{thres}; \\ \forall i, j : c_{ij} \le b_x, \textit{где } f(c_{ij}) = x; \\ \forall x : b_x \le \sum_{i,j} c_{ij} \ . \end{gathered} \quad (3.5)$$

Для целей тестирования используются специальные модули сбора и подготовки фаззинг-тестирования, которые реализованы как самостоятельные программно-аппаратные средства. Такие средства позволяют выполнить поиск данных, направленный на исполнение программных функций для тестирования алгоритмов работы устройства в наиболее интересующих разработчиков режимах.

В результате работы алгоритма фаззинга на каждой итерации формируется матрица ветвлений [110, 111], которая суммирует количество переходов для каждой из вершин графа.

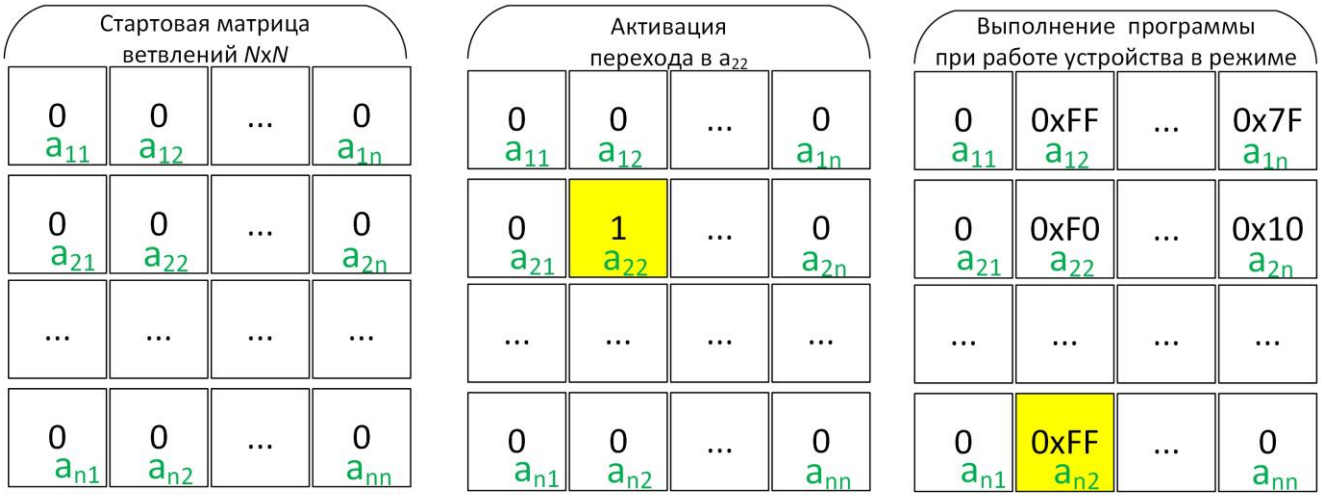



Рисунок 3.1 – Матрица ветвлений для алгоритма фаззинга

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 [98, 114]. Представлена схема организации процесса фаззинга (рис. 3.2). Для этого в алгоритме происходит динамический поиск и добавление новых условных переходов (ветвлений).

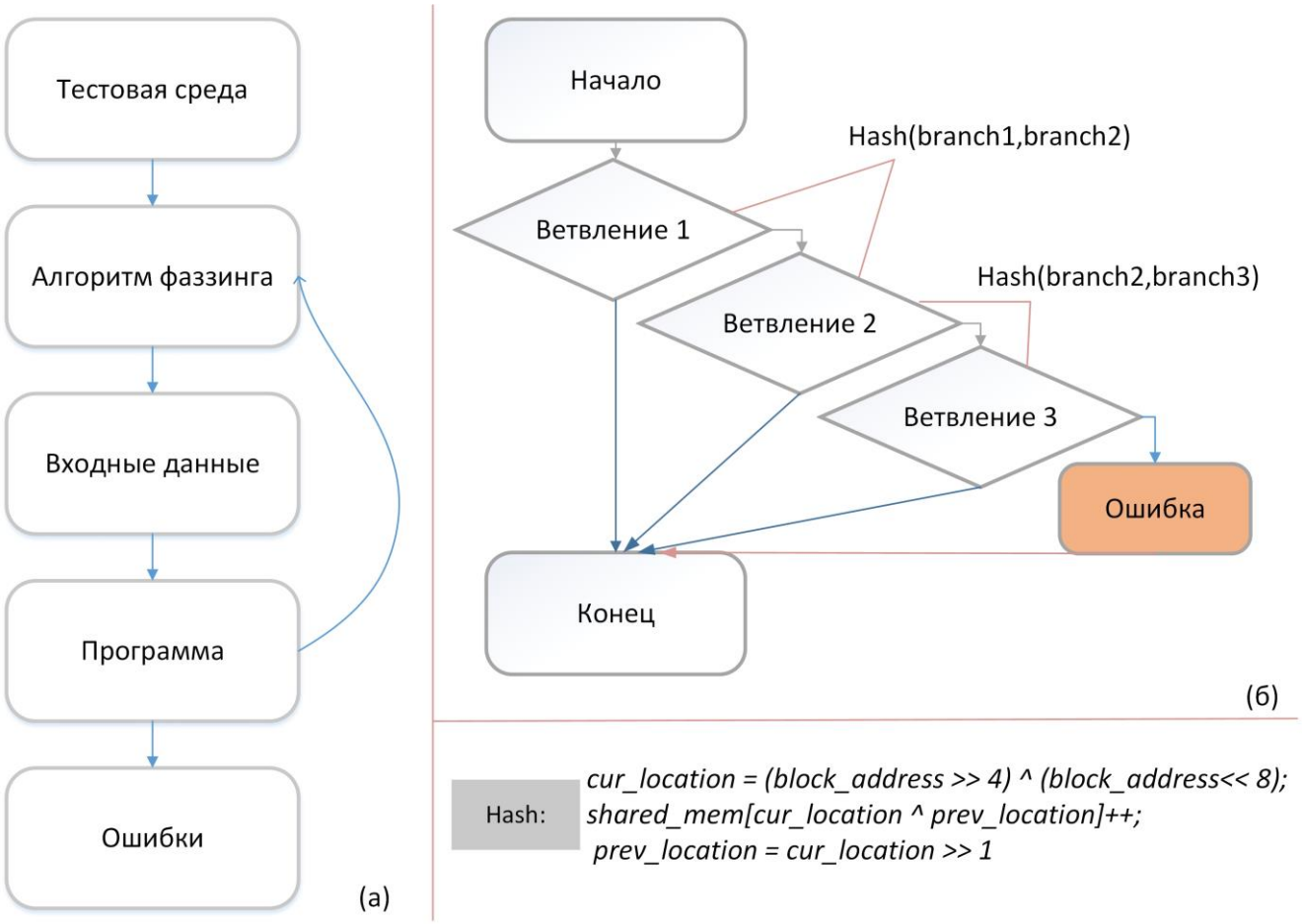



Рисунок 3.2 –Техника фаззинга: а) общий алгоритм, б) контроль ветвлений

В данной работе для микропроцессорного устройства производится сбор статистики работы устройства, и на основе ее анализа формируется подготовка тестовых данных. Поскольку оценка критерия определения объема тестовых испытаний с помощью искажения входных данных в большинстве программ фаззинга [112, 114, 118] производится вручную, то в настоящее время популярны

инструменты, автоматизирующие эти действия. Постановка задачи оптимизации обхода графа программы формулируется следующим образом [105]:

$$\max_{x \in R^n} / \min \mathrm{F(x)} \; при \begin{cases} C_i(x) \geq 0, i \in N; \\ C_i(x) = 0, i \in Q, \end{cases} \tag{3.6}$$

где $F(x)$ – целевая функция, содержащая набор всех достигнутых ветвлений программы; $x$ – данные для полученного пути; $C_i(x)$ – функции ограничений для достижений определенных вершин; $R$, $N$, $Q$ – индексы для ограничений программы, которые определяются для каждой из программ.

Эвристический алгоритм фаззинга часто не позволяет управлять процесса искажения, поэтому достижение интересующих точек работы программы зависит от входных данных и структуры проверяемой программы. Достижение интересующих функций графа осложняется наличием зацикливаний и множественных возвратов, что увеличивает время проведения исследований и снижает эффективность тестирования. На рисунке 3.3 приведен вариант зацикливания программы фаззинга (достижение новых вершин с определенного времени не происходит).

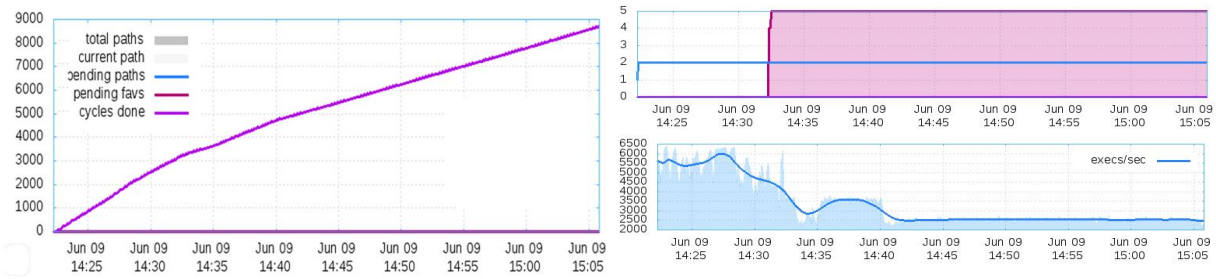



Рисунок 3.3 – Фаззинг программы без использования статистики

При проектировании систем разработчикам требуется решить задачу интеграции техники фаззинга в текущую инфраструктуру тестирования и соединить с модулями, которые в отличие от фаззинга позволяют автоматически получать оценку проводимых испытаний.

Определение объема тестовых испытаний и коррекция работы программы осуществляются сторонними модулями, активно разрабатываемыми в инженерной среде [114]. Тем не менее, использование модулей для микропроцессорного

устройства требует наличия совместных решений по фиксации состояния устройства и контроля аппаратных средств.


Рисунок 3.4 – Применение фаззинга в процессе испытаний

Для выполнения поставленных задач предложен экспериментальный комплекс, который включает в своем составе набор программных и аппаратных компонентов. Компоненты экспериментального комплекса реализуют функции, описанные в таблице 3.1. Функции контроля реализуются в программном обеспечении устройства на разном уровне абстракций (аппаратные драйвера, операционная система, системные утилиты эмуляции, внедрение в пользовательскую программу). Выбор абстракции для контроля исполнения программы устройства производится по результатам экспертной оценки сложности программного обеспечения устройства, которая формируется в результате классического тестирования. Компоненты системы являются законченными аппаратными средствами, что позволяет комбинировать их использование в составе информационного комплекса.

Таблица 3.1 – Компоненты инфраструктуры тестирования

| № | Компонент системы | Описание реализуемых функций |
|---|---|---|
| 1 | Набор микроконтроллеров | Объект испытаний, к которому подключается устройство имитации неисправностей |
| 2 | Рабочая станция баз данных | Получение и обработка данных о результатах выполнения функций программного обеспечения устройства для тестового набора данных |
| 3 | Устройство имитации неисправностей | Верификация инфраструктуры тестирования с использованием тестов системы с известным результатом (сравнение реакции устройства на внесенные данные с требуемой реакцией). Имеется в виду, что если внесли ошибку, которая выявляет сбой при классическом тестировании, то инфраструктура тестирования позволит увидеть, что ошибка произошла. Если ошибки нет, комплекс настроен некорректно |
| 4 | Станция имитации неисправностей | Анализ выполнения программы с помощью эвристических алгоритмов модели организации испытаний (с использованием техники фаззинга) |
| 5 | Статистический программный блок определения отказов /сбоев | Настройка программных средств комплекса для проведения испытаний (инфраструктуры тестирования) для контроля показателей программы при проведении фаззинга |
| 6 | Коммутационный блок JTAG/DMA | Формирование модифицированных данных путем искажения битов данных для достижения наибольшего количества переходов в программе |

На рисунке 3.5 представлена функциональная схема экспериментального комплекса проведения испытаний, которая реализует описанные компоненты. Устройство со средствами диагностики рассматривает контроль ошибок при исполнении программы устройства, а инфраструктура тестирования обеспечивает набор программных модулей для генерации данных и анализа реакции устройства. Опишем блоки функциональной схемы устройства. Программные средства внесения данных и диагностики включают четыре блока:

- первый блок (инструментирование на уровне интерфейсов) использует аппаратный интерфейс *JTAG*/*SWD* для получения и внесения данных в программное обеспечение;

- второй блок (инструментирование на уровне ядра) реализует способ инструментирования на уровне операционной системы (системный уровень) и осуществляет доступ к аппаратным интерфейсам с помощью функции процессора, используя программные средства для процессоров *Intel PT*;
- третий блок (инструментирование на уровне эмуляции ОС) предполагает создание копии операционной системы (эмуляцию) для отслеживания выполнения программы с помощью программ виртуализации *DOCKER* и *QEMU*;
- четвертый блок (инструментирование библиотек) реализует способ использования непосредственно модификации отслеживаемой программы с помощью средств компилятора и динамических библиотек.

Рассмотрим состав и функции инфраструктуры тестирования устройства, основой которой являются программные модули тестирования, реализованные на отдельном (сервисном) компьютере, подключаемом к тестируемому устройству для внесения входных данных и анализа реакции на их внесение.

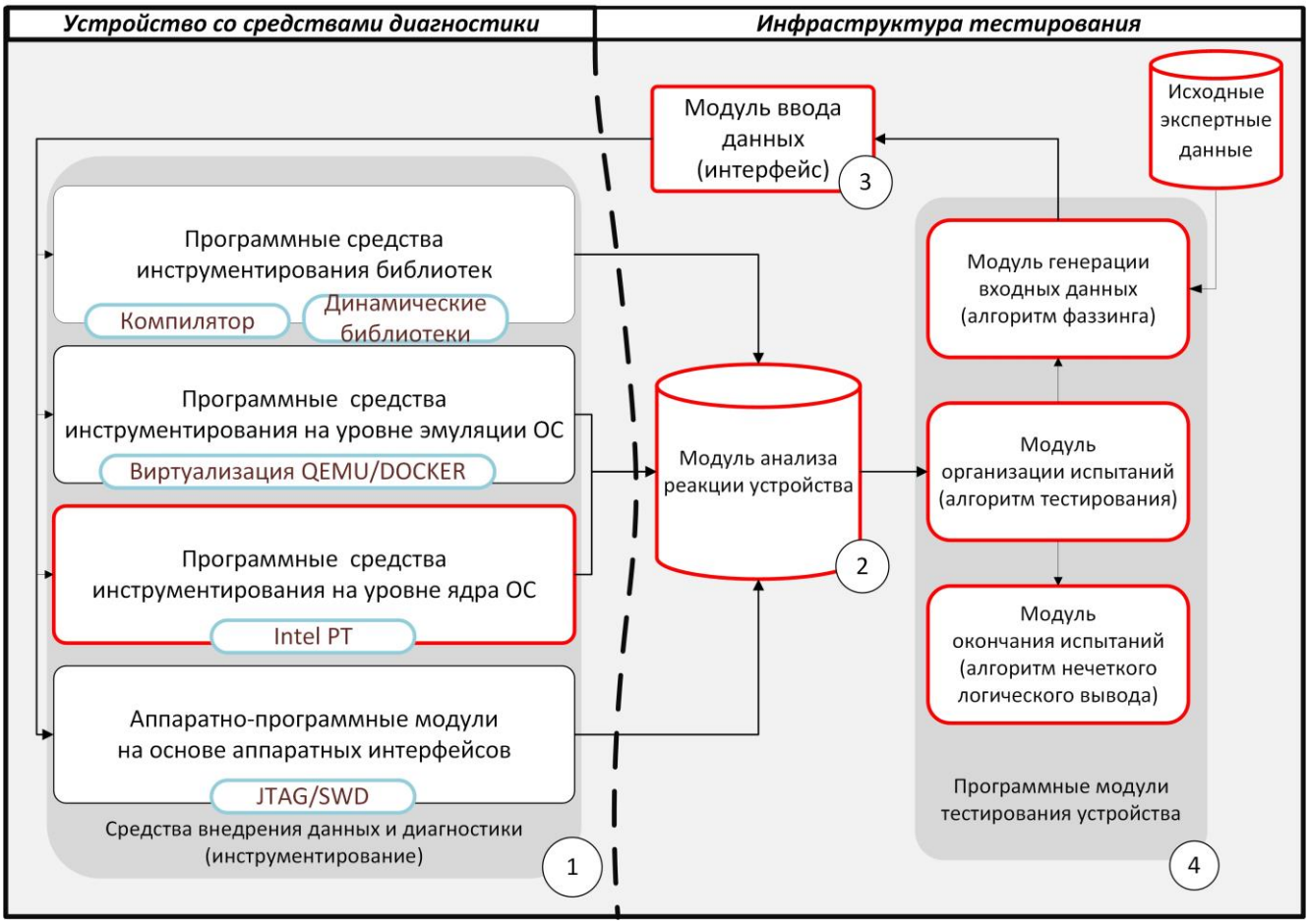



Рисунок 3.5 – Функциональная схема комплекса проведения испытаний

Модуль организации испытаний реализует алгоритм запуска, останова и коррекции испытаний. Для запуска испытаний он использует модуль генерации входных данных, который формирует искаженные данные для внесения в устройство. Модуль окончания испытаний обрабатывает данные от устройства и формирует критерий окончания испытаний.

Модуль ввода данных представляет собой интерфейс обмена для ввода и вывода информации (от средств диагностики) на сервисный компьютер. Модуль анализа реакции устройства реализуется на сервисном компьютере и принимает данные от устройства.

Модуль генерации входных данных с помощью фаззинга является важным компонентом инфраструктуры тестирования, который с использованием эвристического алгоритма генерации данных осуществляет подготовку искаженных входных данных. Подготовка данных заключается в искажении битов экспертных данных в псевдослучайном порядке (с нуля на единицу) и с помощью эвристического алгоритма фаззинга, применяя при этом логические операции *XOR, AND* и их комбинации (а также работы со словарем, перестановки и усечения), с целью задействования большего объема выполняемой программы, что при контроле проявляется как повышение значений метрик фаззинга *PATH_COV и ERR_CNT* (определяются экспериментально).

Устройство со средствами диагностики представлено с точки зрения инфраструктуры для его использования. Рассмотрим подробнее структуру для совместной оценки с помощью представленных средств (рис. 3.6) [48].

Набор тестируемых микроконтроллеров 1 представляет собой несколько устройств, обладающих JTAG-интерфейсами стандарта IEEE 1149.1 или новее, объединенных в JTAG-цепочку посредством последовательного подключения данных устройств. Для проверки набора микроконтроллеров исполняется тестовая программа, результат выполнения которой известен на любой стадии выполнения. При возникновении неисправности система считывает данные с блока с отказного микроконтроллера при отсутствии стандартного сигнала о выполнении операций,

выдаваемого на выход микроконтроллера и фиксируемого другими блоками системы.

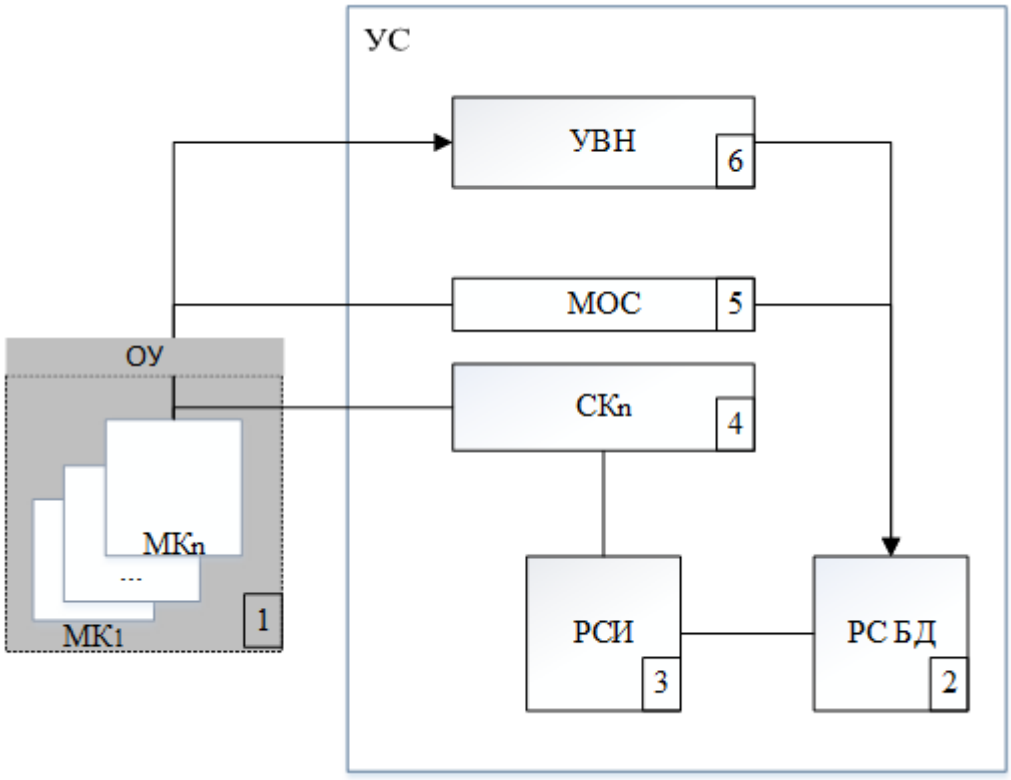



Рисунок 3.6 –Тестовая среда для имитации неисправностей

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 микропроцессора на основе сигнатурного анализа.

Также на рабочей станции запущен экспертный алгоритм, который корректирует зоны применения алгоритма для увеличения скорости моделирования отказов и сбоев. 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 (случайного внесения искажений в данные).

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 на основе математической модели на основе нечеткого логического вывода (рис. 3.7 и 3.8).

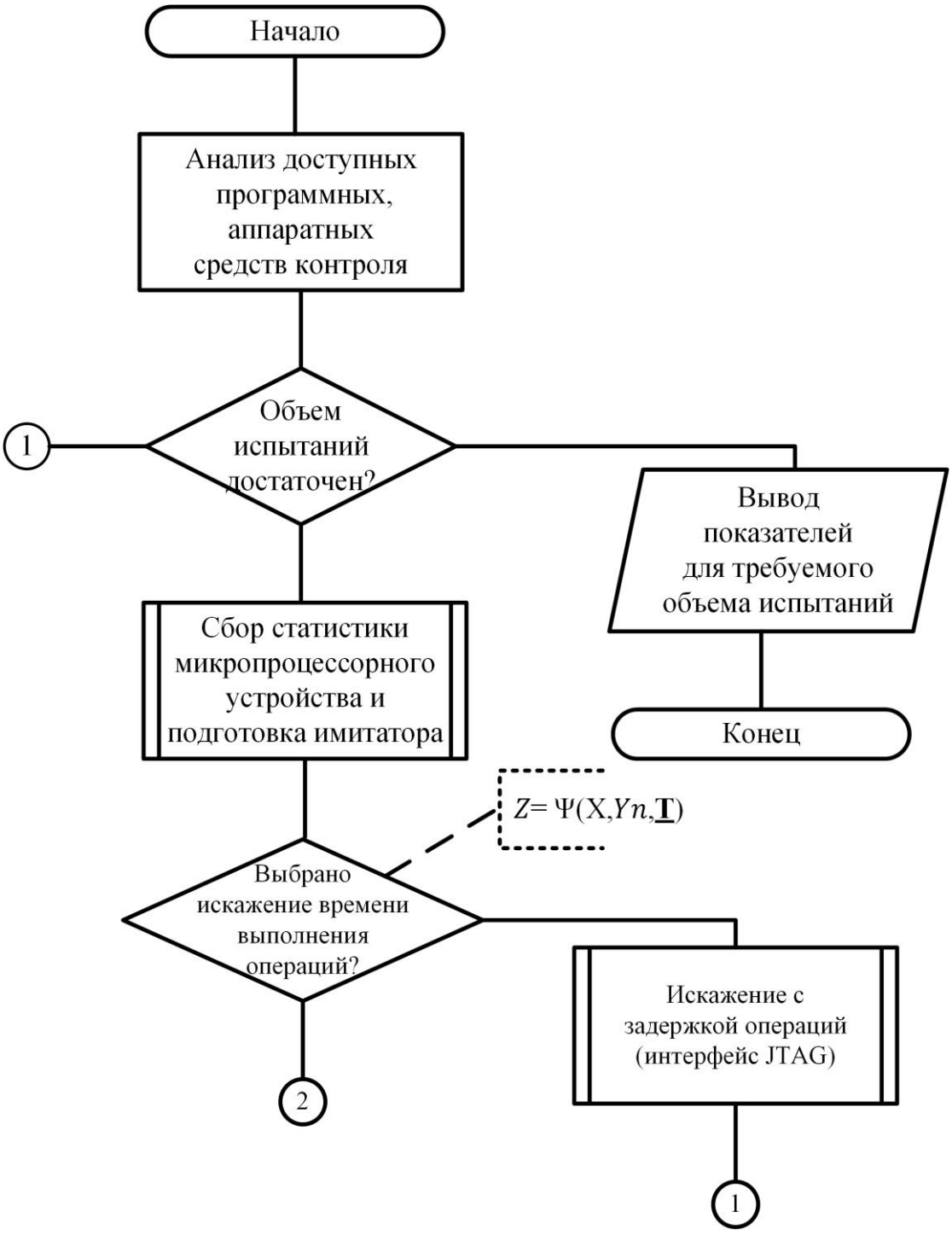



Рисунок 3.7 – Алгоритм анализа проблемных ситуаций устройства

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;

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.

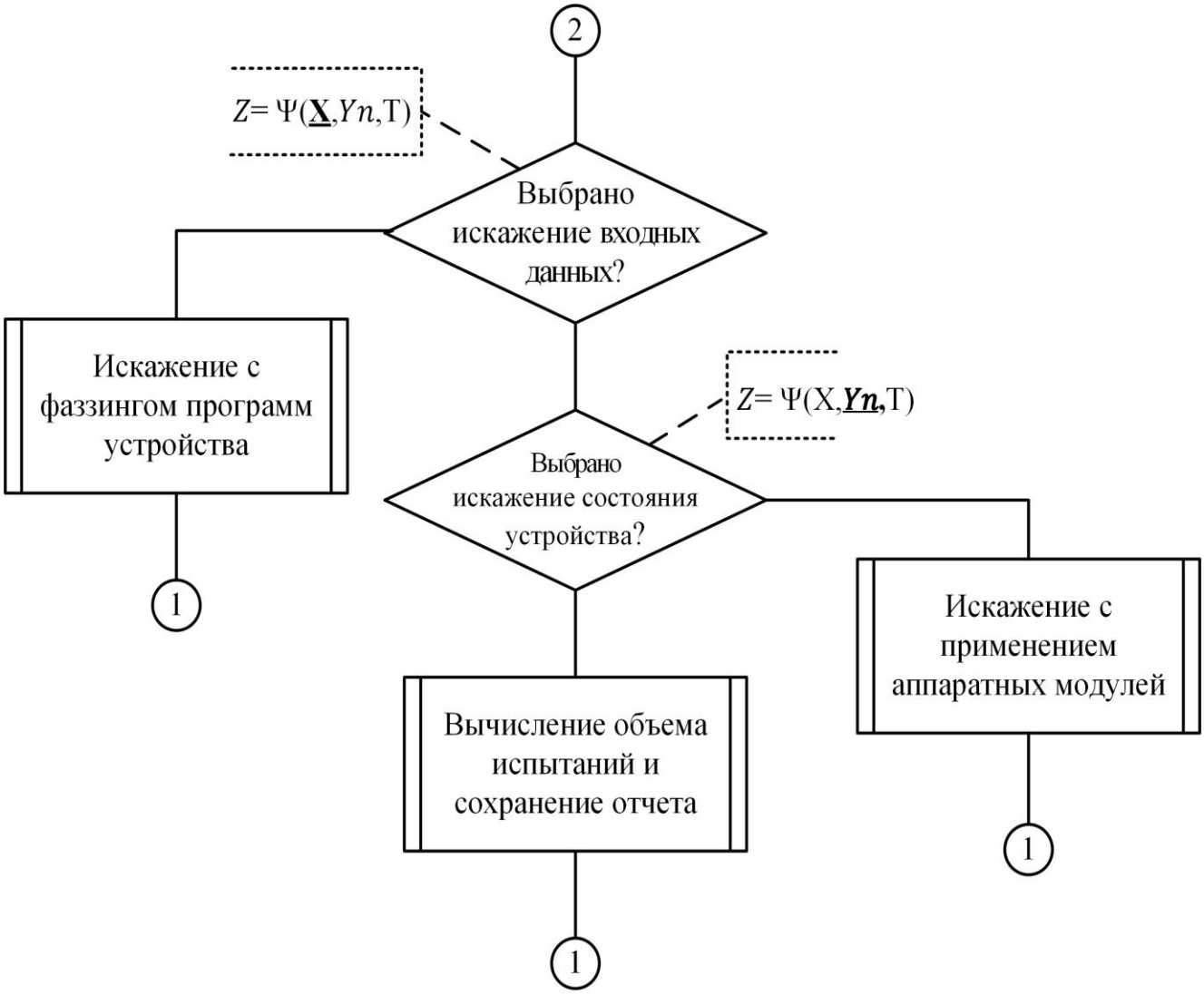



Рисунок 3.8 – Алгоритм организации испытаний (продолжение)

Для осуществления данных пунктов в рамках методики производится ряд вычислений с помощью средств диагностики (на основе JTAG). Это позволяет автоматически диагностику технического состояния для дальнейшего устранения системных ошибок, проявляющихся при загрузке пользовательских программ в реальную аппаратную среду устройства [47, 102] (проявление свойства эмерджентности).

## 3.2 Алгоритм предобработки исходных данных с определением режимов испытания устройства, проверяемых функций и точек контроля

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 точками 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:

$Cmd_R$ – 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;

$\varepsilon_\beta$ – допустимая погрешность оценки среднего времени наработки на отказ;

$T_R$ – время сбора статистики на одну программную функцию.

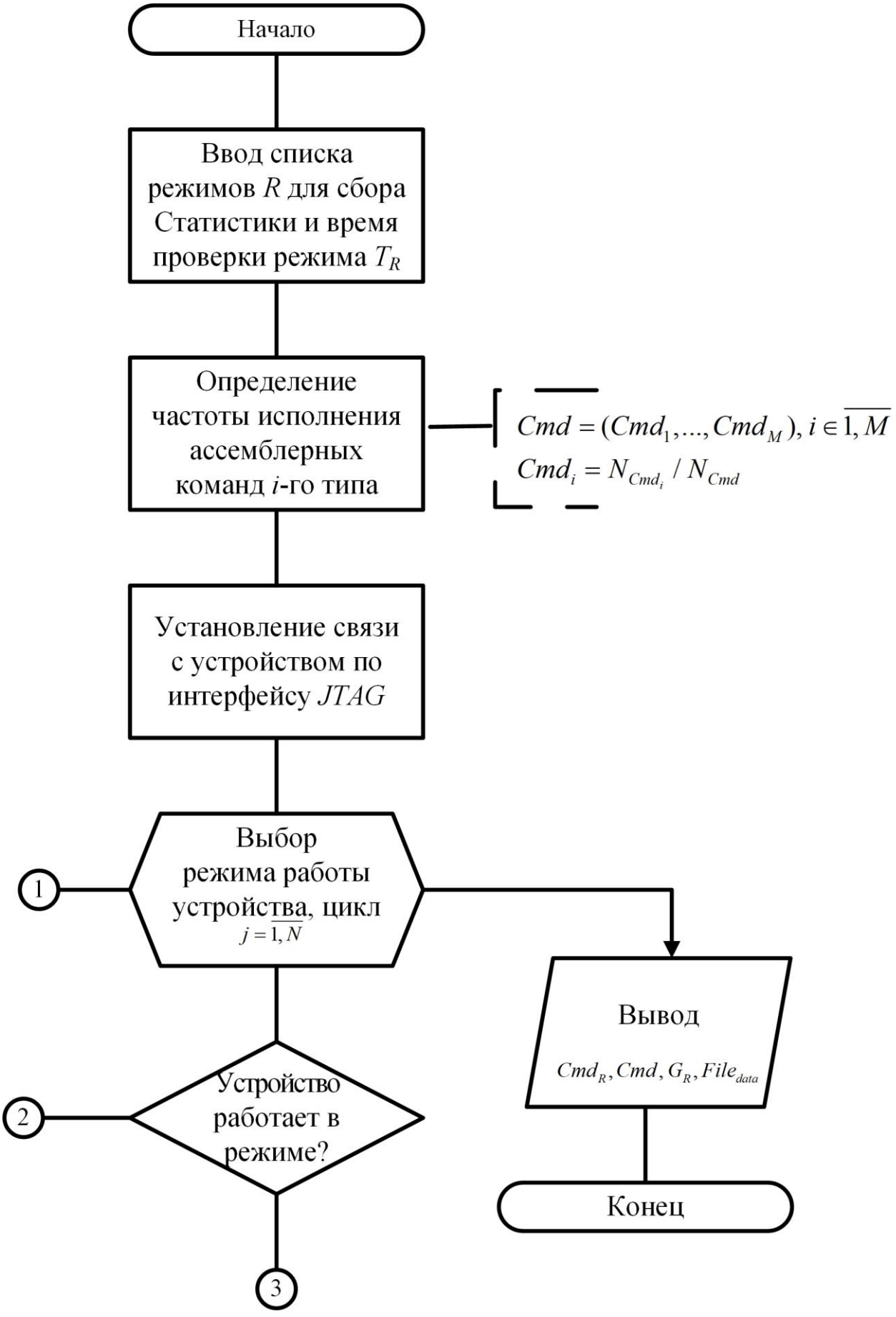



Рисунок 3.9 – Алгоритм предобработки исходных данных

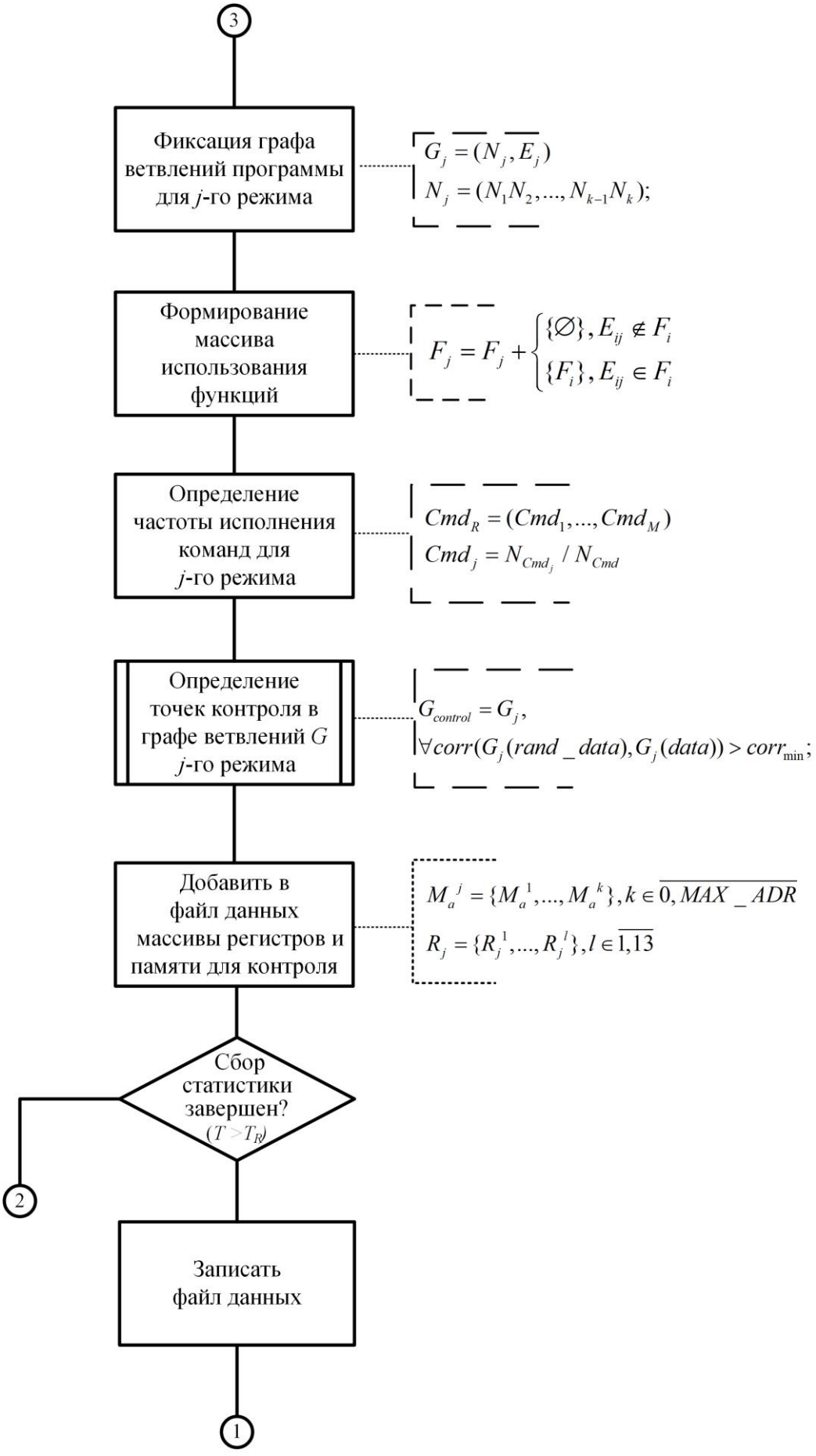



Рисунок 3.10 – Алгоритм предобработки исходных данных (продолжение)

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 указывается в виде файловой директории и передается эвристическому итерационному алгоритму фаззинга.

## 3.3 Алгоритм программы внесения инъекций отказов и сбоев в микропроцессорное устройство на основе техники фаззинга

Тестирование устройства производится путем искусственного внесения ошибок в программное и аппаратное обеспечение проверяемых устройств. В качестве исходных данных используются результаты работы алгоритма предобработки исходных данных. Внесение ошибок производится с помощью искажения данных, основанного на обнаруживаемых ветвлениях в функции. Для внесения отказов и сбоев обозначим основные переменные алгоритма:

$A$ – файловый массив внесенных данных;

$a_j^i$ – данные, переданные в выбранное ветвление *i*-й функции;

$b_j^i$ – ветвление программы *i*-й функции;

$M$ – матрица ветвлений*;*

$F(b_j^i)$ – результат выполнения *i*-й при достижении *j*-го ветвления программы;

$D$ – коэффициент корреляции между временем искажения данных и проявлением отказов и сбоев.

Алгоритм осуществляет последовательное исследование ветвлений программы, допускающих достижения новых ветвлений (путей программы) и в специальных контрольных точках программы для сравнения с шаблоном в заданном режиме. При этом ветвления относятся к конкретной функции. Отказы и сбои фиксируются для всей функции в целом.

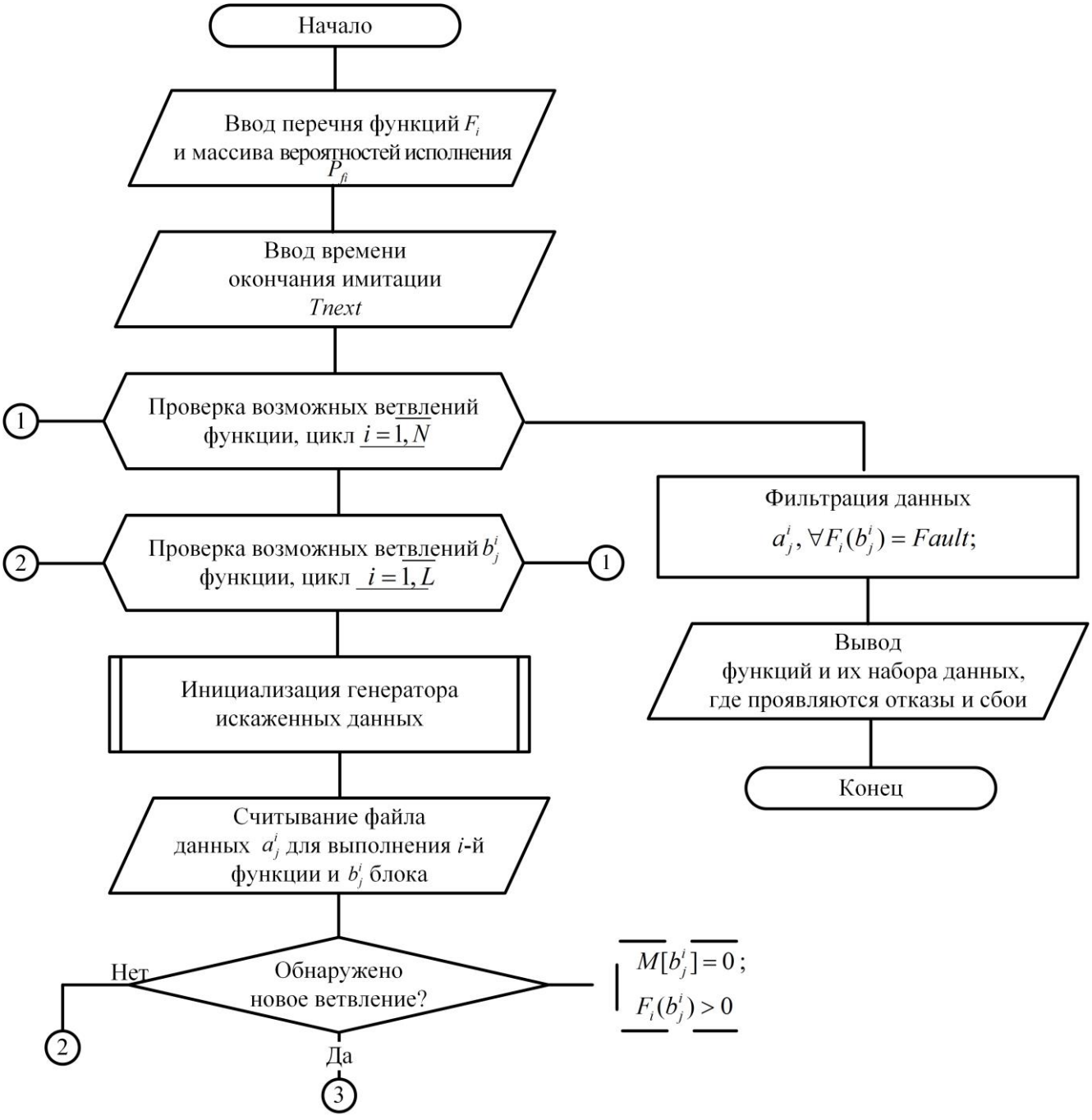



Рисунок 3.11 – Алгоритм программы внесения неисправностей

Алгоритм внесения случайной инъекции данных и алгоритм инъекции данных с подготовленной гипотезой составляют вместе с алгоритмом формирование статистики вызовов группу алгоритмов по инъекции неисправностей.

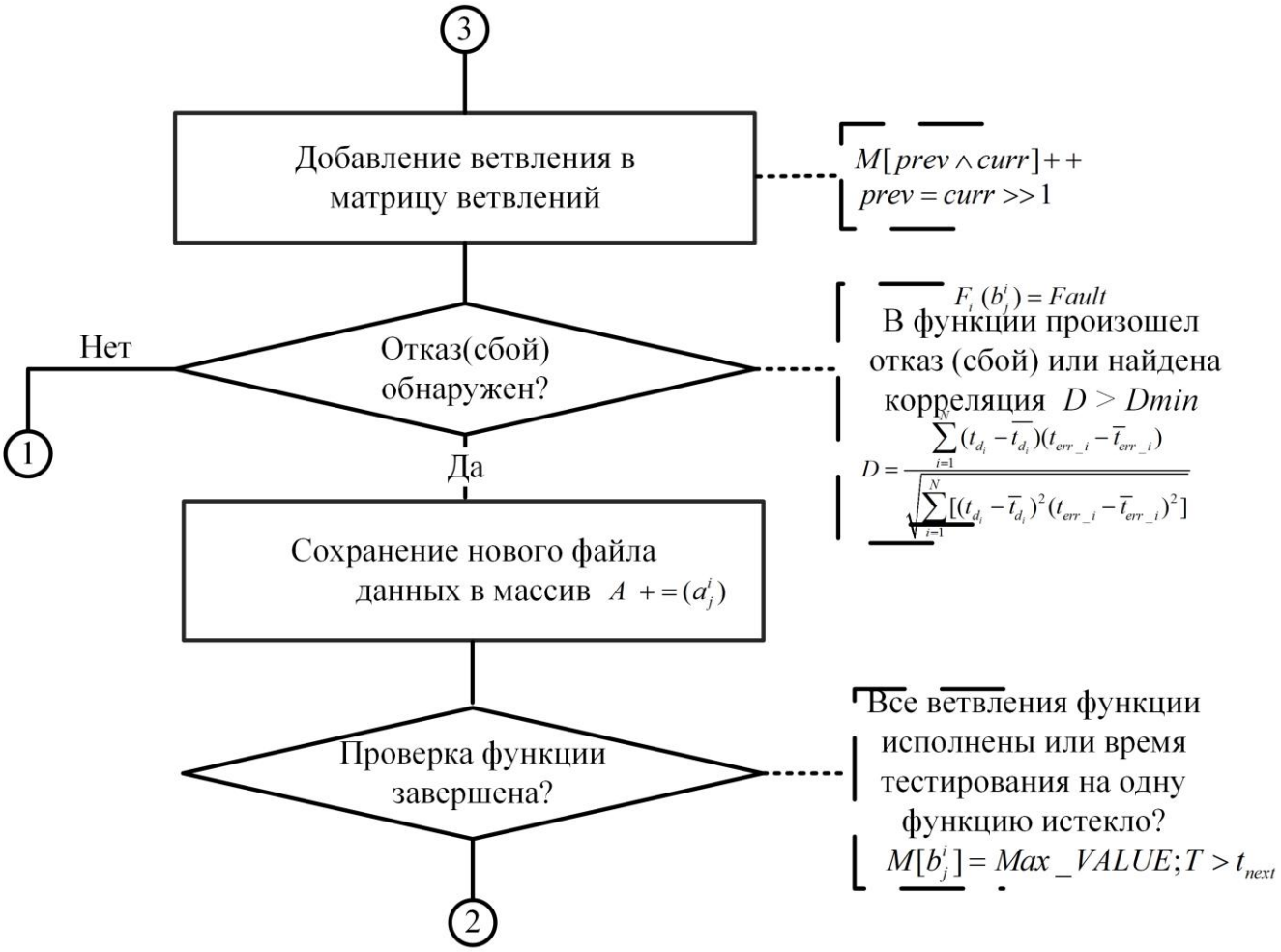



Рисунок 3.12 – Алгоритм программы внесения неисправностей (продолжение)

Тестирование программно-аппаратных устройств с помощью имитации неисправностей регламентируется стандартами IEC 61508 и ГОСТом Р МЭК 61508 и является обязательным требованием при разработке устройств информационных комплексов [62, 63, 119]. Формирование критерия окончания испытаний, позволяющих выявить дефекты сложного микропроцессорного устройства, необходимо осуществлять на основе оценки его основных функций, определяемых техническим заданием. Обычно программа испытаний, формируемая заказчиком проектируемого устройством исходя из набора выполняемых функций, не

гарантирует получение тестового сценария, применимого на практике. Набор испытаний может быть избыточен или недостаточен, так как не учитывает особенности программно-аппаратных средств, реализующих эти функции. Устранение в дальнейшем не выявленных дефектов влечет дополнительные затраты временных и технических ресурсов. Модификация ОЗУ производится синхронно к выполняемым алгоритмам, используя свойства и структуру функций, что позволяет точечно использовать фаззинг.

## 3.4 Алгоритм определения объема тестовых испытаний по имитации неисправностей на основе нечеткого логического вывода

Ключевым преимуществом фаззинга для системного тестирования является поиск аппаратурных неисправностей с помощью подмены данных для интерфейсов или аппаратурных модулей в процессе испытаний, когда реакция устройства непосредственно зависит от аппаратуры. В работе при рассмотрении вопроса тестирования программно-аппаратных устройств основное внимание уделено формированию критерия завершения испытаний, учитывающего не только функции системы, но и особенности программного обеспечения и ключевые параметры аппаратных средств, характеризующие стабильность работы. Предлагается методика оценки выполнения основных функций устройства, реализуемая программой испытаний на основе совместного использования техники фаззинга (для изменения входных данных) и математического аппарата нечеткой логики (для определения момента останова тестирования). В проверяемом информационном комплексе тестированию подлежат устройства, которые предназначены для обмена пользовательской информацией по сети, а также для ее обработки, хранения и последующего анализа. Отклонение от шаблона обработки будет с определенной вероятностью определять наличие ошибок в программном коде [93, 100, 101].

После получения статистики функций требуется сформировать набор инъекций, который используется в экспериментальном анализе устойчивости

устройства к отказам и сбоям. Модификация данных может быть осуществлена при достижении устройством функций, которые выполняются максимально интенсивно в процессе работы выбранного режима устройства. Синхронизация модификации данных и исполняемых функций программы производится с помощью анализа памяти устройства. Инъекция неверных, искаженных данных в различные временные промежутки определяет устойчивость алгоритма. Как было отмечено ранее, для тестирования применяется программная имитация неисправностей с использованием техники фаззинга, позволяющей автоматизировать процесс обнаружения ошибок с помощью генерации искаженных данных на устройствах во время работы. При недостаточном объеме тестовых испытаний вероятность проявления невыявленных, потенциально опасных для системы дефектов значительно повышается, поэтому важно определить критерий окончания тестирования [44, 49, 91].

Поскольку микропроцессорные устройства функционируют, используя сложное программное обеспечение, то предлагается формировать критерий окончания тестирования на основе анализа статистических данных о работе аппаратных и программных средств этих устройств.

Программа испытаний информационного комплекса использует статистические показатели для анализа и принятия решения об окончании или продолжении испытаний. Сбор статистических данных занимает длительное время и выполняется в автоматическом режиме средствами тестирования и диагностики. Начальные данные формируются по экспертным шаблонам. Это означает, что в качестве начальных данных принимаются наиболее часто применяемые при реальной эксплуатации текстовые наборы, передаваемые устройству.

В результате работы программы формируются статистические показатели, характеризующие объем тестовых испытаний, соответствующий требуемой точности оценки среднего времени наработки на отказ (установленной экспертами). Кроме того, предлагается выполнить анализ последующей реакции программного обеспечения устройства, используя нечеткий логический вывод для определения момента завершения проверки выполняемых функций.

Для проведения анализа введем показатели, которые позволят определить момент окончания испытаний:

$\tilde{P}$ – частота отказов и сбоев устройства;

$P_{fi}$ – вероятность исполнения $i$-й программной функции при наступлении отказа или сбоя;

$p$ – вероятность отказов и сбоев устройства;

$K$ – число аппаратных отказов, $N$ – число программных функций;

$\beta$ – точность оценки среднего времени наработки на отказ;

$\varepsilon_\beta$ – допустимая погрешность оценки среднего времени наработки на отказ;

$t_\beta$ – число среднеквадратичных отклонений от центра рассеивания среднего времени наработки на отказ.

Для проведения анализа и определения момента окончания испытаний предлагается использовать следующие четыре показателя [6]. В качестве первого показателя принимается параметр, характеризующий точность оценки среднего времени наработки на отказ $\beta$ (определяется при выполнении программных функций, в процентах):

$$\beta = P(\left|\tilde{P} - p\right| < \varepsilon_\beta), \tag{3.7}$$

где $\tilde{P}$ – частота отказов и сбоев устройства; $p$ – вероятность отказов и сбоев устройства; $\varepsilon_\beta$– допустимая погрешность оценки среднего времени наработки на отказ; $p$ – реальное значение вероятности отказов и сбоев, определяемое при эксплуатации устройства; $P$ – доверительная вероятность, которая определяется как отклонение между реальным значением вероятности возникновения отказов и сбоев и его оценкой по частоте.

Вторым принятым показателем является количество аппаратных отказов при исполнении программных функций за период проведения испытаний $K$. В качестве третьего принимается показатель $I_\beta$, характеризующий вероятность выполнения устройством некорректных действий для данного алгоритма работы (соответствующих отказам и сбоям устройства) во время реализации программных функций. Показатель $I_\beta$ рассчитывается следующим образом:

$$I_\beta \in ([\tilde{P} - t_\beta \sqrt{\tilde{P}(1-\tilde{P})K^{-1}}; [\tilde{P} + t_\beta \sqrt{\tilde{P}(1-\tilde{P})K^{-1}}]), \quad (3.8)$$

где $K$ – количество аппаратных отказов устройства; $t_\beta$ – число среднеквадратичных отклонений от центра рассеивания среднего времени наработки на отказ; $\tilde{P}$ – частота отказов и сбоев устройства. Показатель $K$ фиксируется аппаратными средствами контроля, а показатель $\beta$ устанавливается перед началом испытаний экспертами, которые учитывают требования к устойчивости к отказам и сбоям.

В качестве показателя $D$ принимается коэффициент корреляции между временем внесения ошибок во входные данные и временем наступления отказов и сбоев в процессе проведения испытаний:

$$D = \frac{\sum_{i=1}^{N} (t_{d_i} - \overline{t_{d_i}})(t_{err\_i} - \overline{t}_{err\_i})}{\sqrt{\sum_{i=1}^{N} [(t_{d_i} - \overline{t}_{d_i})^2 (t_{err\_i} - \overline{t}_{err\_i})^2]}}, \quad (3.9)$$

где $N$– количество программных функций; $t_{d_i}$ ($\overline{t_{d_i}}$) – время внесения (среднее время внесения) $i$-й ошибки с момента начала тестирования, мс; $t_{err\_i}$ ($\overline{t_{err\_i}}$) – время проявления (среднее время проявления) $i$-й ошибки с момента начала тестирования, мс. Таким образом, показатель $D$ позволяет определить, есть ли связь между внесением ошибки и ее проявлением или ошибка случайно произошла независимо от внесения неисправности.

Средства диагностики фиксируют показатели с установленным периодом. Следует отметить, что при недостаточном объеме тестовых испытаний вероятность проявления невыявленных потенциально опасных для системы дефектов значительно повышается, поэтому важно определить критерий окончания тестирования. Существуют различные критерии завершения испытаний, обеспечивающие выполнение требований к надежности функционирования системы [1, 41, 67, 73, 75]. Алгоритмы тестирования программно-аппаратных устройств часто используют такой показатель надежности, как наработка на отказ, определяя время проведения испытаний, но не учитывая при этом особенности реализуемого на устройстве программного обеспечения. Кроме того, обычно алгоритмы тестирования не прини-

мают во внимание характер изменения работы программного обеспечения, вызванного изменением входных данных, а также технические особенности проявляющихся отказов и сбоев. Алгоритм выполнения программы испытаний иллюстрирует рисунок 3.13. Для имитации эксплуатации устройства в реальных условиях осуществляется внесение ошибок модулем фаззинга, искажающим входные данные устройства при работе в штатных режимах.

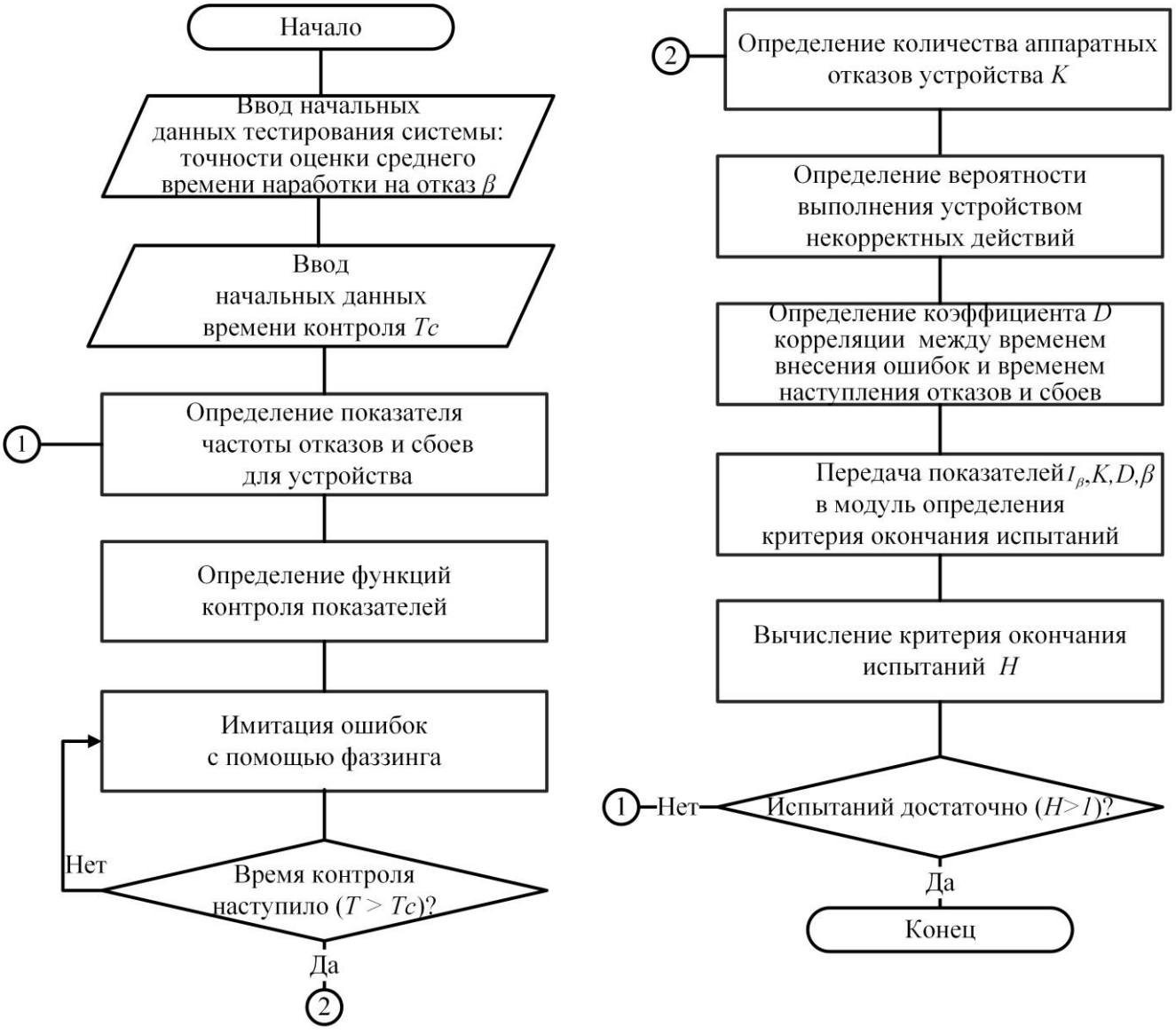



Рисунок 3.13 – Алгоритм выполнения программы испытаний устройства

Искажения входных данных позволяют выявить ошибки при выполнении арифметических и логических команд, и, кроме того, вносимые искажения

последовательности выполнения программы приводят к ошибкам в командах перехода, которые в свою очередь ведут к отказам и сбоям.

Сформированный в результате работы алгоритма критерий окончания испытаний $H$ позволяет определить необходимость дальнейшего проведения испытаний (для выбранного значения точности оценки среднего времени наработки на отказ $\beta$).

Следует отметить, что использование технологии фаззинга для тестирования программ позволяет осуществить автоматический поиск ошибок программного обеспечения устройства с помощью искажения входных данных и последующего анализа реакции устройства. Для проверки работы программного обеспечения устройства с помощью фаззинга использованы две метрики: *PATH_COV* (число путей в программе от общего количества, принимающее значения от 0% до 100%) *и ERR_CNT* (число полученных ошибок), наиболее информативных для оценки результатов применения фаззинга.

Критерий окончания испытаний $H$ формируется при работе специализированных программных и аппаратных средств тестовой инфраструктуры испытательного комплекса [35, 44]:

$$H = \begin{cases} 0, \ \textit{если} \ \sum_{j=1}^{N}\sum_{i=1}^{M} Rule_i(I_\beta, D, K, \beta)\cdot M^{-1} \le 1; \\ 1, \ \textit{если} \ \sum_{j=1}^{N}\sum_{i=1}^{M} Rule_i(I_\beta, D, K, \beta)\cdot M^{-1} > 1, \end{cases} \tag{3.10}$$

где $N$ - число программных функций; $M$ – число продукционных правил для вычисления критерия окончания испытаний заданной функции; $Rule_i(I_\beta, D, K, \beta)$ – результат вычисления критерия окончания испытаний в соответствии с $i$-м продукционным правилом нечеткого вывода (вычисляется на основе показателей). Объем испытаний считается достаточным в тот момент, когда получено первое положительное значение критерия $H$.

Поскольку микропроцессорные устройства функционируют, используя сложное программное обеспечение, то предлагается формировать критерий окончания тестирования на основе анализа статистических данных о работе аппаратных и программных средств этих устройств.

Для определения объема тестовых испытаний микропроцессорных устройств разработана специальная программа тестирования, в которой предусмотрено определение момента завершения испытаний. При этом устанавливается, что объем проведенных испытаний достаточен на основе введенного для этой цели критерия окончания испытаний. После расчетов и фиксации средствами диагностики показатели обрабатываются и вычисляется значение критерия окончания испытаний. Алгоритм вычисления на основе решающих правил нечеткого логического вывода [56, 58] по четырем показателям определяет критерий окончания испытаний. Полученное на выходе данного алгоритма решение является численным значением критерия, по которому определяется, испытания проведены в полном объеме или необходимо их продолжить.

Технический комплекс, предназначенный для проведения испытаний, представляет собой программно-аппаратную систему, используемую для выявления дефектов разработанного устройства перед прохождением приемо-сдаточных испытаний. Комплекс содержит инфраструктуру тестирования, которая настраивается для проверки работоспособности устройства в требуемых режимах работы, а также выполнения штатных функций по техническому заданию для этого устройства. Микропроцессорное устройство, как правило, при функционировании осуществляет регистрацию всех выполняемых операций. Для каждого типа разрабатываемых устройств состав и структура комплекса проведения испытаний отличается, поскольку отличаются схемотехнические решения, интерфейсы и программное обеспечение устройств, поэтому при разработке устройства обычно проектируется и комплекс проведения испытаний, способный проверять функции данного устройства.

Рассмотрим разработанный технический комплекс проведения тестовых испытаний для устройства приема и передачи информации на базе процессоров общего назначения с поддержкой интерфейсов граничного сканирования *JTAG*. Инфраструктура тестирования реализует фиксацию реакции системы на имитацию ошибок. По реакции системы устанавливается, вызваны ли отказ или сбой системы

отсутствием защиты от искаженных данных в программе или вводимая ошибка должна приводить к такому отказу (сбою).

Анализ реакции устройства заключается в периодическом опросе через интерфейс *JTAG* состояния устройства и передачи на компьютер, на котором функционируют программные модули тестирования устройства для расчёта показателей $I_{\beta}$, $\beta$, $K$, $D$. Полученные на тестируемом устройстве показатели передаются модулю окончания испытаний, где определяется численное значение критерия окончания испытаний $H$. Модуль окончания испытаний реализован как отдельная подпрограмма, и его работа основана на аппарате нечеткой логики; модуль выполнен в среде MATLAB / Fuzzy Logic Toolbox [15, 16, 17].

В построенной системе на входы нечёткого контроллера поступают четыре сигнала: первый сигнал - это вероятность исполнения *i*-й функции в момент наступления отказа или сбоя ( $p_i$ ). Вторым входным сигналом является количество фиксаций аппаратных отказов и сбоев при исполнении *i*-й функции, (лингвистическая переменная *K*). Третьим входным сигналом является установленная точность вычислений для системы (лингвистическая переменная $\beta_i$ ). Четвертым входным сигналом является установленная корреляция времени между искажением в *i*-й функции и проявлением отказов и сбоев (лингвистическая переменная *D*).

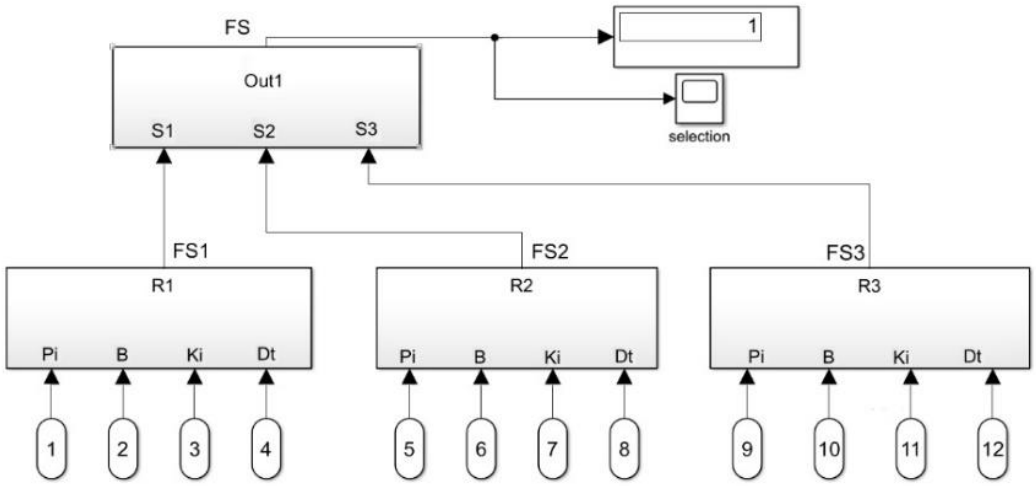



Рисунок 3.14 – Определение критерия окончания тестирования устройства

На выходе нечёткого контроллера выдается значение критерия окончания испытаний (лингвистическая переменная *H*), на основе которого формируется решение об окончании испытаний. В соответствии с полученным значением имитация для микроконтроллерного устройства либо продолжается, либо останавливается с фиксацией текущих значений контролируемых показателей. Модуль в процессе тестирования на основе показателей работы устройства формирует значение критерия окончания испытаний. Для этого полученные в результате имитации неисправностей численные значения показателей используются как входные данные для алгоритма работы нечеткого логического вывода. На рисунке 3.15 представлена схема вычисления критерия окончания тестовых испытаний (в модуле определения окончания испытаний) программно-аппаратного устройства. Для реализации алгоритма нечеткого логического вывода входные переменные показателей $I_\beta$, $\beta$, $K$, $D$ переводятся в значения нечетких лингвистических переменных блоками фаззификации *MF*1 - *MF*4. При выполнении фаззификации число, принадлежащего множеству действительных чисел, представляется в виде нечеткого числа (от англ. *fuzzy* – нечеткий) [11].

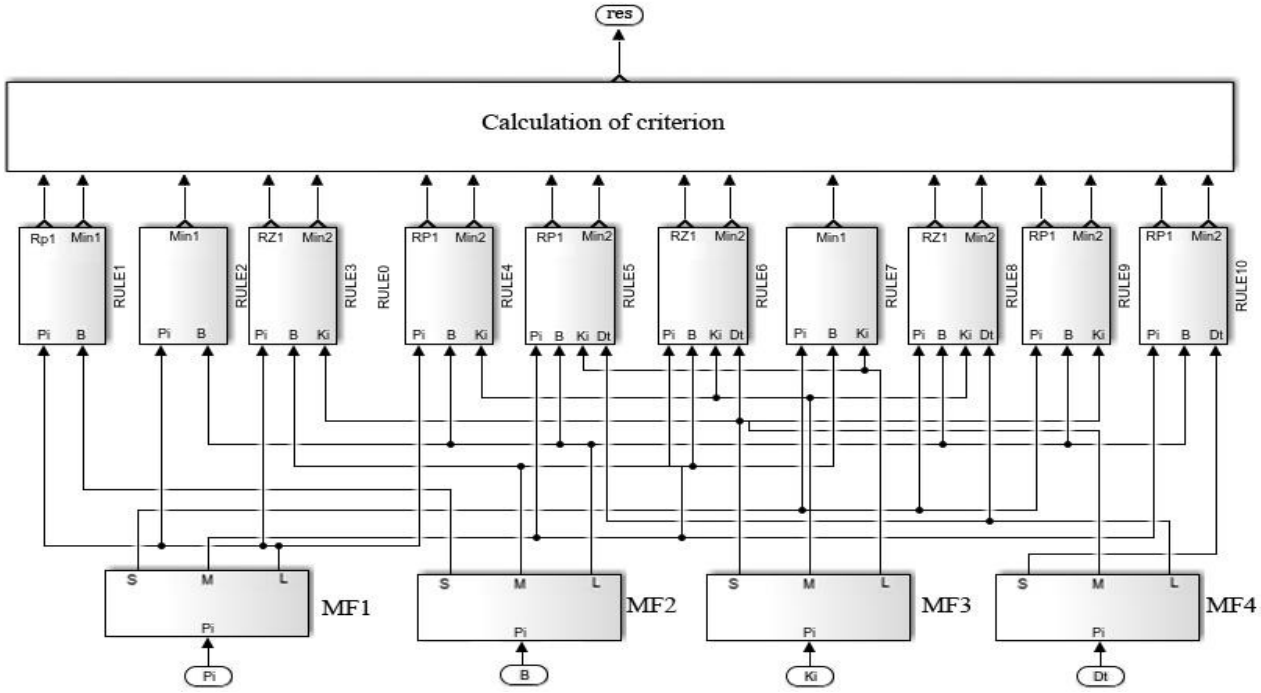



Рисунок 3.15 – Схема вычисления критерия окончания испытаний нечеткого логического вывода

В соответствии с принятой логикой алгоритма, база знаний нечеткого регулирования включает в себя десять нечетких продукционных правил. При реализации продукционных правил для получения нечеткого множества выходной переменной выполнялись операции импликации и агрегирования. Для каждого правила базы знаний импликация выполнялась с помощью операции минимума.

П1:**ЕСЛИ** $p_i$ = L **И** $b_d$ = S, **ТО** R = P;

П2:**ЕСЛИ** $p_i$ = L **И** $b_d$ = L, **ТО** R = N;

П3:**ЕСЛИ** $p_i$ = L **И** $b_d$ = M **И** $k_j$ = S, **ТО** R = Z;

П4:**ЕСЛИ** $p_i$ = L **И** $b_d$ = M **И** $k_j$ = M, **ТО** R = P;

П5:**ЕСЛИ** $p_i$ = M **И** $b_d$ = M **И** $k_j$ = M **И** $d_t$ =L, **ТО** R = P;

П6:**ЕСЛИ** $p_i$ = M **И** $b_d$ = M **И** $k_j$ = M **И** $d_t$ =M, **ТО** R = Z;

П7:**ЕСЛИ** $p_i$ = S **И** $b_d$ =L **И** $k_j$ = L, **ТО** R = N;

П8:**ЕСЛИ** $p_i$ = S **И** $b_d$ =L **И** $k_j$ = M **И** $d_t$ =L, **ТО** R = Z;

П9:**ЕСЛИ** $p_i$ = S **И** $b_d$ =L **И** $k_j$ = S, **ТО** R=P;

П10:**ЕСЛИ** $p_i$ = S **И** $b_d$ =S **И** $k_j$ =S, **ТО** R=Z.

Рисунок 3.16 – Правила нечеткого логического вывода

Результат логического вывода по всей базе знаний находится агрегированием нечетких множеств с использованием операции максимума. Для каждого решающего правила составляется схема, реализуемая с помощью средств Matlab (рис 3.17). Для фаззификации входных переменных (показателей $x=(I_\beta, \beta, K, D)$) диапазоны их изменения разбиваются на лингвистические термы (на основе экспертных оценок). При этом для каждого показателя используется три терма значений: *S* – малое, *M* – среднее, *L* – большое. Для каждого из термов строится функция принадлежности $\mu(x)$ переменной *x* этому терму.

Для задания внутренних лингвистических термов (*M* – среднее) входных переменных использована симметричная гауссова функция принадлежности (*gaussmf*), формируемая в соответствии с выражением

$$\mu(x) = e^{-(x-c)^2/(2\sigma^2)}, \qquad (3.11)$$

где параметр *c* задает модальное значение функции, а $\sigma$ – ширину.

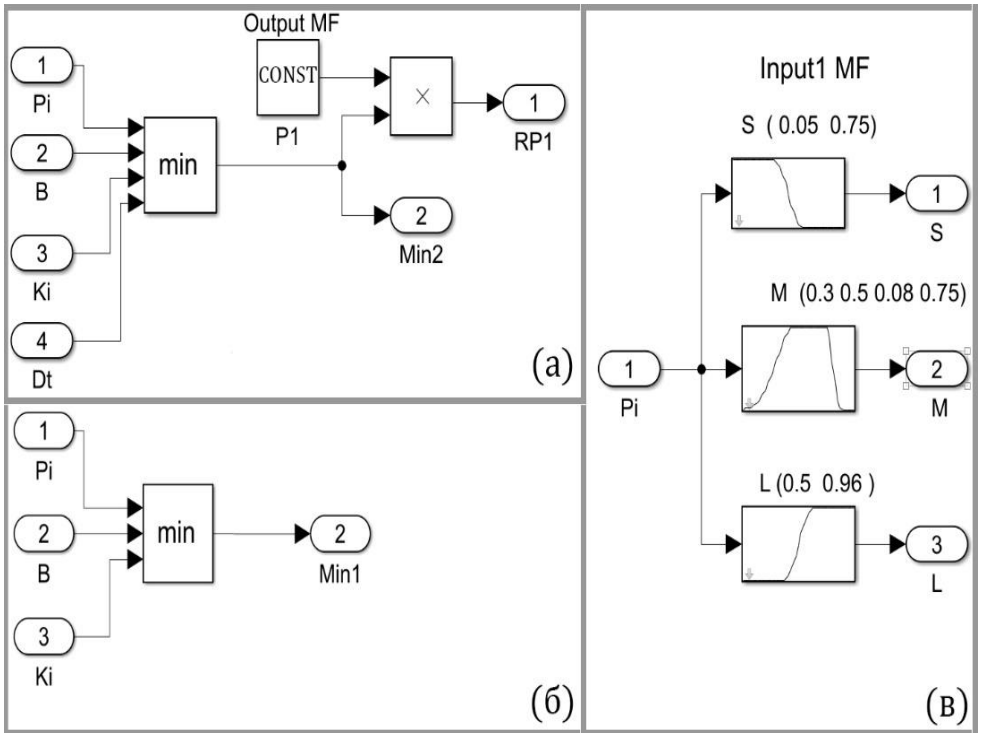



Рисунок 3.17 – Модель испытаний устройства: а) правило нечеткого вывода для положительного решения; б) правило нечеткого вывода для отрицательного решения; в) подсистема фаззификации

Для задания крайних термов (*S* – малое и *L* – большое) используются сигмоидальные функции принадлежности (*sigmf*), которые определяются в соответствии с выражением:

$$\mu(x) = (1 + e^{-a(x-c)})^{-1}, \text{ где } a < c. \quad (3.12)$$

Параметр *c* определяет координату точки перегиба функции, а коэффициент *a* задает наклон функции в этой точке. При этом *a*>0 используется для *S*-образной функции принадлежности, а при *a*<0 реализуется *Z*-образная функция принадлежности.

Заключение каждого правила задается в виде четкого числа критерия окончания испытаний *H*, которое реализовано как одноэлементное нечеткое множество с точечной функцией принадлежности. Диапазон изменения переменной *H* также разбивается на три терма: *PL* – положительное, *PZ* – нейтральное, *PN* – отрицательное (в соответствии с принадлежностью значения критерия этим термам принимается решение об окончании испытаний).

Далее производится определение четкого числа критерия окончания испытаний $H$ путем выполнения процедуры дефаззификации (обратного преобразования нечетких переменных в четкие). Вычисление критерия окончания испытаний $H$ осуществляется путем определения взвешенного среднего в соответствии с выражением [12, 107]:

$$H = \sum_{i=1}^{m} \mu(H) H_i \left(\sum_{i=1}^{m} \mu(H)\right)^{-1} . \qquad (3.13)$$

Здесь $H_i$ – значение выходной переменной для $i$-го терма с единичным значением степени принадлежности; $\mu(H)$ – степень принадлежности к этому терму; $m$ – число термов. Для осуществления нечеткого логического вывода используется совокупность нечетких правил, реализующих на основе нечетких переменных (для показателей $I_\beta$, $\beta$ ,$K$ ,$D$) вычисление критерия окончания испытаний $H$ по формуле (5). Правила нечеткого вывода $RULE1$ – $RULE10$, которые реализуются для каждой из контролируемых программных функций, формируются в соответствии с таблицей 3.2.

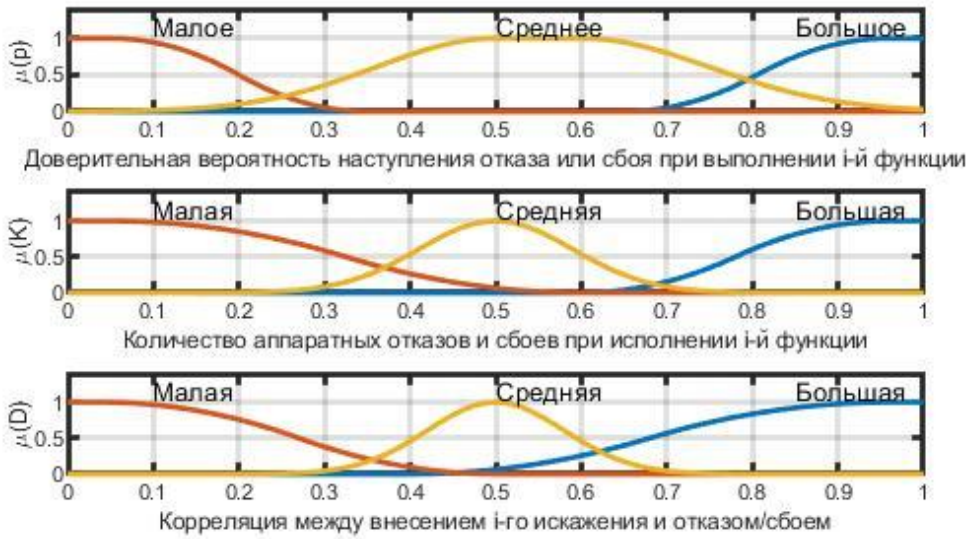



Рисунок 3.18 – Функции принадлежности для термов входных переменных, полученные в результате экспертных оценок

Отбор программных функций для контроля показателей производился таким образом, чтобы внесение ошибок при их выполнении было наиболее значимо для работы устройства.

Таблица 3.2 – Правила определения критерия окончания испытаний *H*

| Показатель $I_\beta$ | Показатель *K* | | |
|---|---|---|---|
| | *S* | *M* | *L* |
| При показателе $\beta = S$ | | | |
| *S* | *PZ* | - | - |
| *M* | - | - | - |
| *L* | - | - | *PL* |
| При показателях $mo = M$, $D = M$, $D = L$ | | | |
| *S* | - | - | - |
| *M* | - | *PZ* | *PL* |
| *L* | *PZ* | *PL* | |
| При показателях $\beta = L$, $D = L$ | | | |
| *S* | *PL* | *PZ* | *PN* |
| *M* | - | - | |
| *L* | - | - | *PN* |

При тестировании рассматривались два режима работы устройства: режим приема данных *R*1 и режим обновления данных *R*2. Для режима приема данных выбраны три функции $N = 3$ ($f_1$ - функция приема, $f_2$ - функция установки соединения, $f_3$ - функция настройки). Для режима обновления данных выбраны пять функций $N = 5$ ($f_1$ – функция приема, $f_2$ – функция установки соединения, $f_3$ – функция настройки, $f_4$ – функция проверки данных, $f_5$ – функция обновления данных).

Поскольку выходные данные от устройства передаются в модуль окончания испытаний неравномерно по времени (в связи с задержками в интерфейсе передачи информации), то в рамках программы испытаний предусмотрено, что вычисленные показатели передаются для формирования критерия окончания испытаний за установленный временной интервал [15, 19].

Распределение процессорного времени между программными функциями для режимов работы отображено в столбце «Загрузка процессора» определяет время на исполнение процессором выбранной функции.

Таблица 3.3 – Результат сбора информации о работе функций программы

| Тестируемая программа для трех функций | | | |
|---|---|---|---|
| Номер | Загрузка процессора | Адрес ОЗУ | Точка вызова функции |
| 1 | 10.57% | 3523065 | [.] *main* |
| 2 | 18.48% | 2811321 | [.] *saveDataStorage* |
| 3 | 32.82% | 677691 | [.] *reloadSysHand* |
| Тестируемая программа для пяти функций | | | |
| 1 | 10.57% | 3523065 | [.] *main* |
| 2 | 18.17% | 2811321 | [.]*saveDataStorage* |
| 3 | 32.37% | 677691 | [.]*reloadSysHand* |
| 4 | 1.01% | 5532105 | [.] *testBrokenServ* |
| 5 | 16.15% | 3514688 | [.] *reloadDriver* |

Параметр «Адрес ОЗУ» – размещение функции в оперативной памяти тестируемого устройства. Столбец таблицы «Точка вызова функции» определяет название функции, которое встречается в исходном коде программы и фиксируется средствами инструментирования программного кода.

## 3.5 Выводы по главе

1. Разработана структура комплекса для проведения тестовых испытаний, которая позволяет использовать устройство для имитации неисправностей для автоматического поиска ошибок в программно-аппаратном обеспечении с использованием техники фаззинга. Показано, что применяемая стратегия фаззинга предусматривает на каждом этапе изменения данных (перестановка битов и байтов, арифметические операции) анализ реакции программы на внесенные искажения. Это дает возможность обеспечить полноту исследования корректности работы программных функций в реальных режимах работы устройства.

2. Предложен способ и алгоритм предобработки исходных данных с определением режимов испытания устройства, проверяемых функций и точек контроля, которые обеспечивают ускорение фаззинга за счет подготовки данных для исследования графа ветвлений на основе наиболее часто используемых

функций. Такой способ позволяет осуществить эффективный контроль наиболее уязвимых мест ПО.

3. Модифицирован алгоритм внесения неисправностей с помощью техники фаззинга и разработан алгоритм внесения инъекций отказов и сбоев в микропроцессорное устройство. Отличие предлагаемого алгоритма от существующих заключается в использовании интерфейса JTAG при анализе реакции на проимитированные неисправности в процессе обхода графа ветвлений программы (с помощью генерируемой матрицы ветвлений). Основное преимущество модифицированного алгоритма обусловлено использованием предобработанных статистических показателей в качестве входных данных, что позволяет повысить полноту исследования основных программных функций и ускорить выполнение их проверки.

4.Разработаны метод и алгоритм определения объема тестовых испытаний на основе нечеткого логического вывода. Предложен критерий определения объема тестовых испытаний микропроцессорных устройств. Критерий, основанный на предварительной статистической обработке экспериментальных данных о выполняемых функциях в совокупности с алгоритмом принятия решения на базе нечеткого логического вывода, позволяет оценить достаточность объема испытаний.

5.Создана математическая модель для обоснования критерия, учитывающего предварительно полученные характеристики устройства: число аппаратных отказов, вероятность ошибок при выполнении программных функций, а также корреляцию между временем внесения ошибок во входные данные и временем выявления неисправностей.

# 4 РЕЗУЛЬТАТЫ ИСПЫТАНИЙ И ЭКСПЕРИМЕНТАЛЬНЫХ ИССЛЕДОВАНИЙ УСТРОЙСТВА ИНФОРМАЦИОННОГО КОМПЛЕКСА

В **четвертой главе** приведены результаты испытаний микропроцессорного устройства и экспериментальные исследования в инфраструктуре тестирования, которая содержит устройство для имитации неисправностей.

## 4.1 Организация экспериментальных исследований для устройства информационного комплекса

Экспериментальные исследования устройства заключались в обнаружении отказов и сбоев при тестировании программно-аппаратного (микропроцессорного) устройства и обнаружении ошибок программ и аппаратных неисправностей, которые являлись причиной отказов и сбоев. В таблице 4.1 приведен список [23, 26] вносимых искажений, которые найдены с помощью имитатора неисправностей.

Таблица 4.1 – Вносимые искажения для микропроцессорного устройства

| Номер | Виды искажений | Точки внедрения и практическая реализация |
|---|---|---|
| **REG** | Искажения в работе при изменении регистров в процессе выполнения | Запись значений по адресам регистров |
| *R-01* | Модификация регистров интерфейсов обмена микроконтроллера | Интерфейсы параллельной и последовательной передачи-приема |
| *R-02* | Модификация регистрового файла процессора | Регистры общего назначения, стека, адреса возврата, состояния процессора |
| *R-03* | Модификация регистров ввода-вывода микроконтроллера | Управление уровнем, подтяжкой, режимом, доступностью вводов-выводов |
| **DATA** | Искажение в работе при изменении данных процессора | Изменение значения данных для выполняемых операций |

| Номер | Виды искажений | Точки внедрения и практическая реализация |
|---|---|---|
| *D-01* | Модификация ОЗУ процессора | Перезапись значений до и после выполнения критических функций |
| *D-02* | Модификация данных на интерфейсах устройства | Перехват и промежуточные операции при передаче |
| *D-03* | Модификация данных во flash-памяти процессора | Изменение инструкций программы в процессе выполнения |
| **TIME** | Искажение времени исполнения функций программы | Нарушение допустимых границ отклика и зацикливание |
| *T-01* | Модификация времени задержек при работе с памятью | Проверка памяти на изменение времени доступа |
| *T-02* | Нарушение времени работы с интерфейсами | Зацикливание и пропуск данных |
| *T-03* | Модификация времени выполнения алгоритмов | Проверка устойчивости алгоритмов при нештатных задержках |

Стенд для тестирования [59, 60] состоит из устройства, АРМ оператора и устройства имитации неисправностей и анализа реакции устройства на внесенные искажения (рис. 4.1). В режиме внесения искажений был проведено обнаружение режимов, где проявлялись отказы и сбои. После этого был запущен процесс работы алгоритма предобработки и последующая имитация неисправностей. Пунктирным контуром обозначена структура тестируемого устройства.

Тестируемое устройство подключено к автоматизированному рабочему месту оператора. АРМ оператора подключается через интерфейс сканирования блоков устройства (*JTAG*) к выводам модулей для считывания состояния [38]. Для целей нашего исследования данная функция использована как возможность искажать данные и состояние регистров в процессе работы устройства.

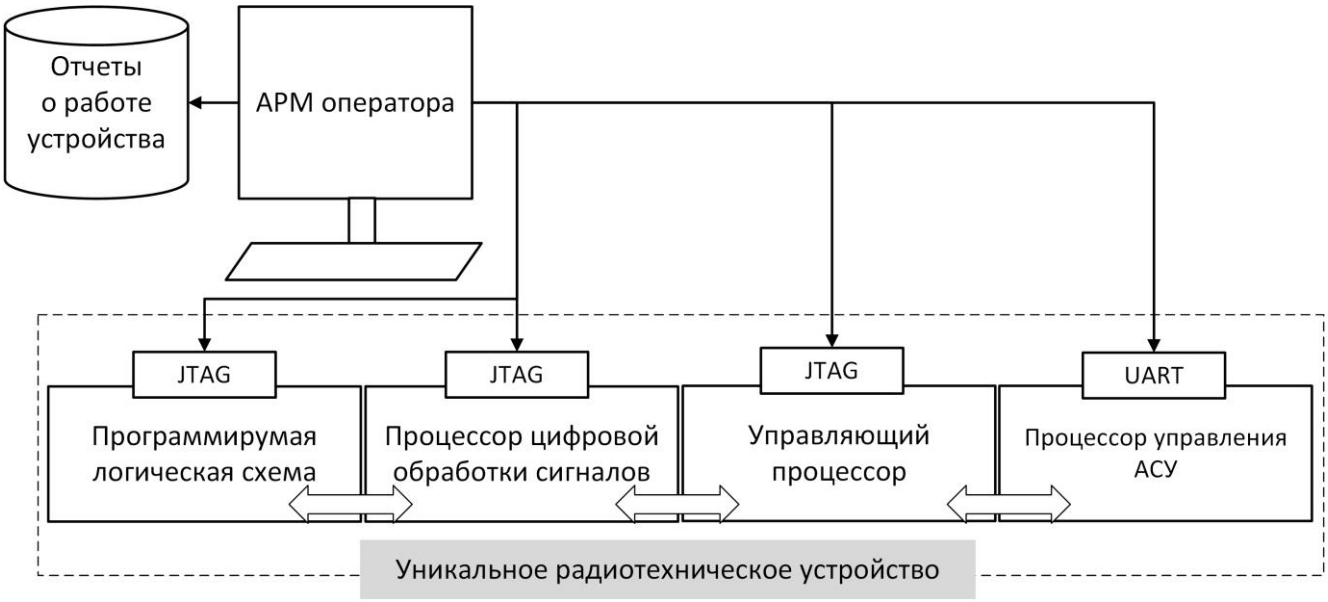



Рисунок 4.1 – Рабочее место экспериментальных исследований

Верификация имитатора неисправностей производилась с помощью двух режимов организации испытаний: псевдослучайного искажения данных в процессе работы устройства и искажения данных на основе полученных сведений о статистике использования функций и ассемблерных команд в режимах работы. Результаты проверки устройства имитации приведены в таблице 4.2 и 4.3.

Таблица 4.2 – Результаты псевдослучайного искажения данных

| № | Количество тестов | Количество реальных отказов/сбоев | Количество фиксаций отказов/сбоев | Процент обнаруженных отказов |
|---|---|---|---|---|
| 1 | 150 | 3 | 0 | 0% |
| 2 | 400 | 10 | 1 | 10% |
| 3 | 600 | 28 | 2 | 7% |
| 4 | 900 | 35 | 1 | 2% |
| 5 | 1200 | 41 | 3 | 7% |
| Средний процент обнаружения отказов и сбоев тестов | | | | **5%** |

Для проверки использовались версии программного обеспечения с известными ошибками, которые выявлялись при работе программного блока имитатора. Проверка инъекции неисправностей с использованием статистической

информации об ассемблерных командах, исполняемых функциях в режимах работы устройства позволило выявить дополнительно 19% ошибок.

Таблица 4.3 – Результаты статистического искажения данных

| № | Количество тестов | Количество реальных отказов/сбоев | Количество фиксаций отказов/сбоев | Процент обнаруженных отказов |
|---|---|---|---|---|
| 1 | 150 | 9 | 3 | 30% |
| 2 | 400 | 39 | 6 | 15% |
| 3 | 600 | 60 | 10 | 16% |
| 4 | 900 | 65 | 3 | 15% |
| 5 | 1200 | 78 | 15 | 19% |
| Средний процент обнаружения отказов и сбоев тестов | | | | **19%** |

Было проведено исследование статистики выполнения ассемблерных команд для режимов работы устройства и выполнения основных функций.

Таблица 4.4 – Результаты сбора статистики об ассемблерных командах

| № | Название команды | Частота использования команд для режимов – *R1/R2/R3* |
|---|---|---|
| 1 | Mov | 60% / 56% / 68% |
| 2 | Call | 12%/ 24% / 12% |
| 3 | Jmp | 12%/ 11%/ 7% |
| 4 | Xor | 10%/ 6%/ 3% |
| 5 | And | 5% / 2%/ 1% |

Таблица 4.5 – Результаты сбора статистики об основных функциях

| № | Программная функция | Область памяти, *%* | Количество аппаратных модулей, шт. | Диапазон времени выполнения, *мс* |
|---|---|---|---|---|
| 1 | tx_data() | 12 | 3 | 100 – 300 |
| 2 | reload() | 7 | 2 | 5 – 1200 |
| 3 | save_data() | 22 | 1 | 20 – 10000 |
| 4 | rx_data() | 16 | 2 | 90 – 50000 |
| 5 | prepare_data() | 23 | 0 | 75 – 1700 |

Полученная в таблицах 4.4 - 4.5 информация использована при работе алгоритма предобработки статистических данных. В результате классического

тестирования были выявлен ряд ошибок. В таблице 4.6 приведены классы ошибок, которые приводили к отказам и сбоям в процессе эксплуатации устройства. Ошибки в процессе тестирования были зафиксированы средствами диагностики.

Таблица 4.6 – Выявление ошибок ПО с помощью классического тестирования

| № | Тип ошибки | Причина появления | Количество Обнаруженных ошибок при классическом тестировании, шт. |
|---|---|---|---|
| 1 | Ошибки типов | Отсутствие системы тестирования в проекте | 10 |
| 2 | Ошибки переходов программы | Ошибки архитектуры программ | 1 |
| 3 | Ошибки управления ресурсами | Утечка динамической памяти | 7 |
| 4 | Ошибки аппаратных модулей | Потеря данных или наводки | 3 |
| 5 | Ошибки интерфейса | Неизученные свойства внешних модулей | 10 |
| Общее число ошибок | | | 31 |
| Отказы и сбои без фиксаций | | | 17 |

Как видно из таблицы 4.6, обнаружение ошибок с помощью классического тестирования фиксирует *31* проявление ошибок, при этом в *17* отказах и сбоях так и не удалось установить причину возникновения. Были использованы аппаратные модули для поиска аппаратных отказов, которые проявлялись в результате испытаний.

Модули аппаратной фиксации отказов и сбоев были верифицированы на микропроцессорных устройствах, которые обладают известными аппаратными проблемами. В результате проверки отказы и сбои были верифицированы, кроме модуля контроля вносимых изменений. Использование аппаратных модулей совместно с классическим тестированием новых результатов не дало.

Таблица 4.7 – Результаты эксперимента

| *Тип отказа/сбоя* | *Детектирование отказа* | *Пропуск отказа/сбоя* |
|---|---|---|
| Сбой тактовой частоты шины (Модуль 2) | да | да |
| Отказ корректности данных (Модуль 4) | да | нет |
| Отказ нарушения прав доступа к памяти (Модуль 5) | да | да |
| Отказ времени Выполнения (Модуль 6) | да | да |
| Модуль контроля вносимых изменений (Модуль 8) | нет | нет |

## 4.2 Описание программных средств испытательного комплекса

Разработка программы устройства осуществляется на рабочей станции испытательного комплекса. Программа, разработанная на языке C/C++ транслируется в исполняемый файл (формат файла – *elf*), который с помощью средств отладки загружается средствами отладки и диагностики в микропроцессорное устройство.

Файл исходного кода программы может быть модифицирован как в процессе выполнения на устройстве, так и в процессе разработки (рис. 4.2). Модификация данных производится в процессе использования имитатора в секции *text*, *data*, *rodata*. Программные средства инфраструктуры реализуют алгоритм управления устройством имитации неисправностей и позволяют выполнять тестирование за счет анализа реакции программы в определенные временные интервалы или по возникновению аппаратных событий. При этом выполняется считывание доступных регистров и оперативной памяти. Это позволяет производить комплексный анализ состояния устройства на рабочей станции.

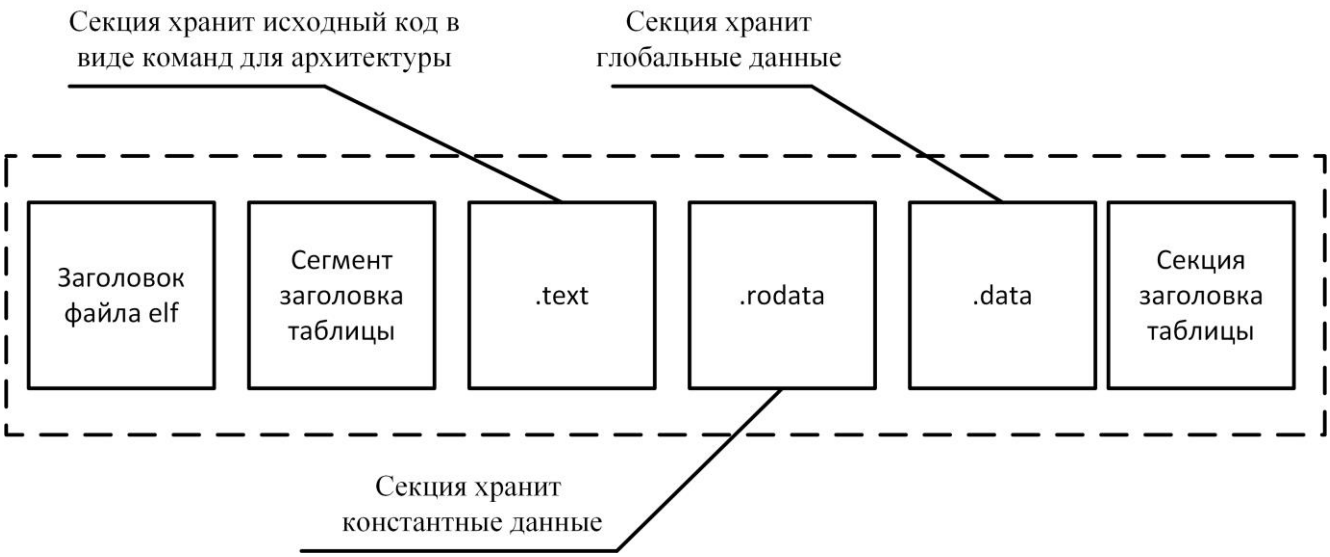



Рисунок 4.2 – Файл исходного кода

Среды программирования (STM32 Workbench, Eclipse и другие) содержат возможность добавлять контрольные точки («breakpoint») для сбора шаблонной информации о выполнении программных функций. Микропроцессорные модули используют типичную схему подключения (общую для встраиваемых систем) с аппаратной поддержкой событий загрузки текущих значений регистров и оперативной памяти.


Рисунок 4.3 – Структура программно-аппаратных компонентов отладки

Программа разработки через плагины позволяет назначить точку останова на любую инструкцию программного модуля. Точке останова соответствует ассемблерной инструкции внутри программной функции. Поиск адреса инструкции, который должен быть модифицирован, производится с помощью файла компоновки для микропроцессорного устройства (*map*-файл). Файл содержит описание размещения функций во *flash*-памяти микроконтроллера.

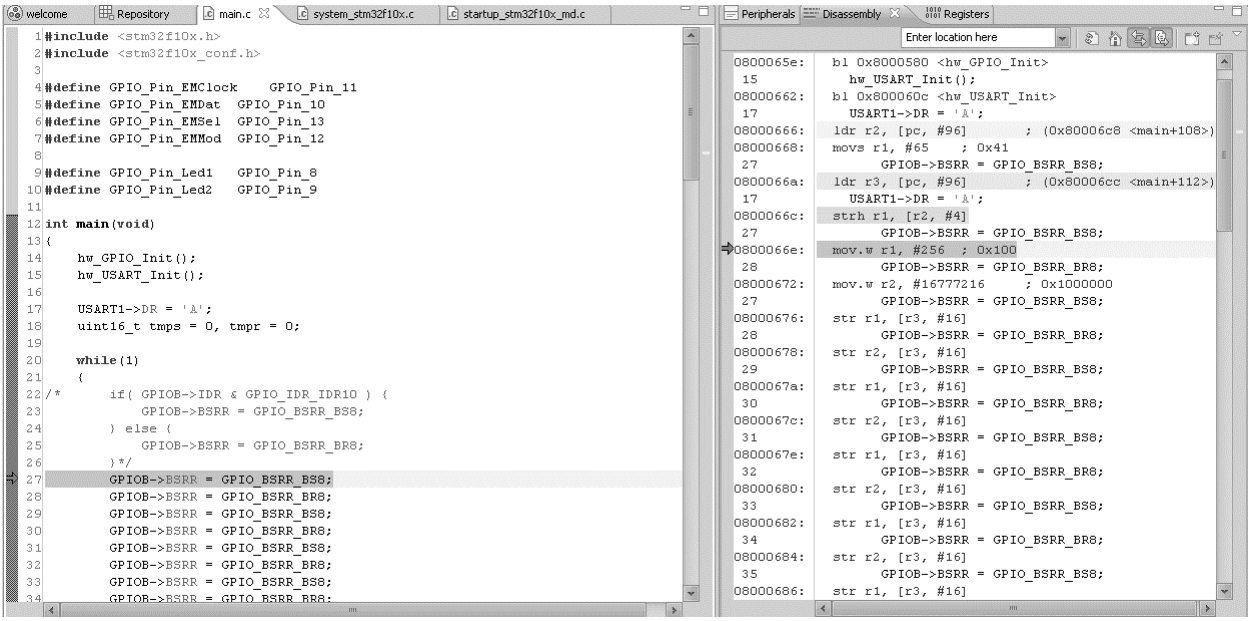



Рисунок 4.4 – Среда разработки с установленной точкой останова

После установки точки останова JTAG-клиент отправляет сигнал на останов процессора (если выполнение запущено), и JTAG-сервер перезаписывает участок flash-памяти микроконтроллера, в которой инструкция для остановки. На рисунке 4.5 показана рабочая среда с установленной точкой останова.


Рисунок 4.5 – Файл размещения программы во *flash*-памяти

Для поиска инструкции вызова используются объектные файлы модулей программы, которые формируются в результате процесса компиляции проекта в исполняемый файл формата *elf*.

```
largo% readelf -i 1 hello.o
0x00000000  pushl       %ebp
0x00000001  movl        %esp,%ebp
0x00000003  pushl       $0x0
0x00000008  call        0x08007559
0x0000000d  addl        $4,%esp
0x00000010  movl        %ebp,%esp
0x00000012  popl        %ebp
0x00000013  ret
```

Рисунок 4.6 – Поиск ассемблерных инструкций

Искажение входных данных для исследуемого файла производится с помощью внедрения в регистровый файл (*R01-R03*) и ОЗУ (*D01-D03*). Подготовленная последовательность данных передается по интерфейсу JTAG и помещается в память устройства или регистры. Последовательная проверка корректности инструкции с помощью известных результатов работы в точках ветвлений программы после сбора информации о статистике использования функций в режимах приема и передачи реализуется в виде алгоритма, который в каждый момент работы проверяет результаты инструкций и подготавливает новый вид искажения, который зависит от текущего типа команды изучаемой функции. Это позволяет осуществлять сразу несколько видов искажений (рис. 4.7):

– искажение с целью нарушения команды пересылки предполагает изменение данных с потов ввода-вывода, модификацию адресов;

– искажение с целью нарушения команды логики предполагает изменение условия ветвления программы для проверки корректности работы алгоритма с данными;

– искажение с целью нарушения команды перехода предполагает модификацию адреса перехода для проверки корректности работы программы;

– искажение с целью вызова (маскирования) прерывания предполагает пропуск или несвоевременную реакцию программы устройства на аппаратные события или события интерфейсов;

– искажение операций ввода-вывода предполагает искажение данных интерфейсов для проверки корректного функционирования в совместных программных модулях.

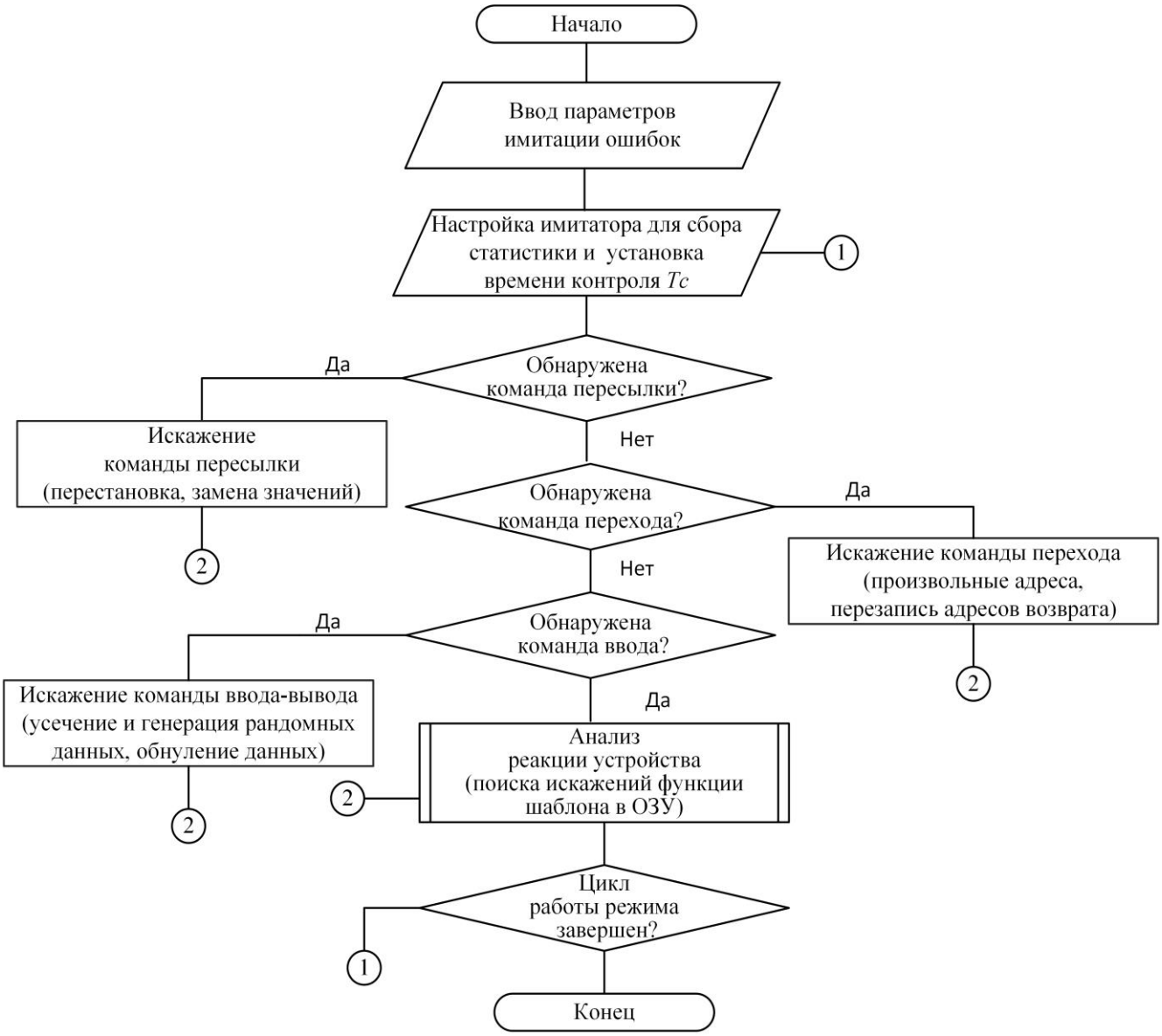



Рисунок 4.7 – Алгоритм внесения искажений *REG/DATA* - типа в программу

Инъекция отказов и сбоев производится с помощью модификации параметров, которые указаны в таблице 4.7. Таблица представляет набор установленных функций библиотек, которые используются для трассировки, поиска и модификации участков программы в автоматическом и полуавтоматическом режимах. Для проведения инъекции требуется установить

место в программе, время, содержимое инъекции [51]. Модификация в соответствии с таблицей производится с помощью трех основных вариантов: использование синхронизации от внешнего импульса, генерируемого портом ввода-вывода по возникновению контролируемого события, остановка процессора, модификация текущих исполняемых данных и возобновление работы процессора; использование порта внутрикристального отладчика *JTAG* и выполнение действий предыдущего варианта с синхронизацией по аппаратной точке останова процессора; генерация отказов и сбоев без привязки к программно-аппаратной работе устройства.

Таблица 4.8 – Основные процедуры оценки программы в псевдокоде

| Псеводокод | Описание |
|---|---|
| **print** T<br>**while** (FUNC != MAIN)<br>**PREV** FUNC<br>**SEARCH** MEM <FUNC><br>**GOTO** MEM<br>**PRINT** <FUNC, ARG> | 1.1 трассировка стека вызов<br>**FUNC** – функция<br>**MEM** – адрес функции<br>**ARG** – список аргум. |
| set_binary_tree(NAME FUNC)<br>**out** get_binary_tree(FUNC) | 1.2 Поиск<br>глубины вложенности |
| Buffer[N][LIST_ARG_OUT] = {...}<br>**while** (FUNC != FAILT_FUNC)<br>**For** (j = 0 to N)<br>**NEXT** FUNC<br>**SAVE** ALL_REGS<br>Buffer[j][LIST_ARG_OUT] = {FUNC OUTPUT}<br>**CALC** MEDIAN VALUE FOR BUFFER FUNC<br>For (j = 0 to N)<br>***DEBUG AND TRACE*** {MODELE FAULT FOR MEDIAN} | 1.3 Модификация параметров<br><br>**FAILT_FUNC** – сбойная функц.<br>**MEDIAN** – медианное знач. буф.<br>**ALL_REGS** – состояние регистров |

Трассировка программы была выполнена с помощью средств отладчика с поддержкой компиляторов GDB/Clang.

### 4.3 Результаты имитации и обнаружения неисправностей

При проведении тестовых испытаний работы устройства информационного комплекса и имитации ошибок с помощью технологии фаззинга определены значения контролируемых показателей устройства. Результаты расчёта показателей приведены в таблице 4.9.

Таблица 4.9 – Результат тестирования устройства информационного комплекса

| *Количество функций программы* | Показатели | | | | Критерий окончания испытаний *H* | *Загрузка процессора*, % | *Время, час* | *Найдено ошибок* |
|---|---|---|---|---|---|---|---|---|
| | $I_\beta$ | $\beta$ | *D* | *K* | | | | |
| Прием данных (режим *R*1) | | | | | | | | |
| *N = 3* | 0.5 | 0.8 | 0.5 | 0.3 | 1 | 67,36 | 12 | 24 |
| | 0.7 | 0.8 | 0.4 | 0.7 | 1 | | | |
| | 0.6 | 0.8 | 0.4 | 0.8 | 0 | | | |
| Обновление данных (режим *R*2) | | | | | | | | |
| *N = 5* | 0.4 | 0.8 | 0.17 | 0.1 | 0 | 78,27 | 12 | 18 |
| | 0.7 | 0.8 | 0.4 | 0.6 | 1 | | | |
| | 0.8 | 0.8 | 0.35 | 0.7 | 1 | | | |
| | 0.8 | 0.8 | 0.76 | 0.9 | 0 | | | |
| | 0.5 | 0.8 | 0.17 | 0.32 | 1 | | | |

На основе анализа показателей для режимов приема данных *R*1 и обновления данных *R*2 сформированы интервалы изменения показателей, внутри которых обеспечиваются требования к критерию окончания испытаний (для выбранного значения точности оценки среднего времени наработки на отказ $\beta$). На основе полученных значений критерия окончания испытаний *H* принимается решение о необходимости прекращения испытаний.

На основе экспертных оценок, а также исходя из условий эксплуатации системы и требований технического задания принималось значение требуемого показателя $\beta$=80%, остальные параметры рассчитывались по результатам работы средствами инфраструктуры тестирования в рамках программы испытаний. Как

можно видеть, критерий окончания испытаний устройства в значительной степени зависит от наличия аппаратных отказов (значения параметра *K*), что учитывается при формировании решения о завершении испытаний. Кроме того, при формировании критерия окончания испытаний учитывается корректность работы программного обеспечения.

Анализ значения параметра *D* позволяет выявлять изменения при выполнении алгоритма работы (отклонения от работы по штатному алгоритму) под влиянием вносимых в устройство искаженных данных.

Результаты тестирования для программного обеспечения устройства иллюстрирует рисунок 4.8. Столбцы значений показателей содержат допустимые интервалы, выделенные серым цветом. Для каждой проверяемой функции принимается решение об окончании тестирования на основе величины степени принадлежности критерия к множеству значений, соответствующих множеству положительных решений. Общий критерий окончания тестирования для всего устройства определяется на основе результатов, полученных при работе модуля окончания испытаний для всех функций, и это решение представляется на дисплей. Значение критерия окончания тестирования иллюстрируется цветом: светлый цвет – достаточно для завершения испытаний, серый цвет – не определено, темный цвет – недостаточно для завершения испытаний. Как можно видеть, критерий *H,* который соответствует завершению испытаний, соответствует объединению допустимых интервалов для всех показателей (серый цвет). Светло-серым и темным цветом, соответственно, показаны значения критерия *H,* для которых испытания должны быть продолжены.

Следует отметить, что при увеличении числа функций возрастает и количество ограничений для имитации неисправностей, поэтому время на проведение испытаний значительно возрастает. Как показали результаты исследования, выбранные показатели работы устройства представляется возможным применять для тестирования и других устройств из этой же партии при серийном производстве (идентичных в части программно-аппаратных решений),

используя предлагаемый критерий для окончания испытаний при серийном тестировании.

Кроме того, в результате исследований установлено, что с ростом числа выполняемых функций каждый из четырех показателей по отдельности (в связи со сложностью определения диапазонов их изменения для большого количества функций) становится недостаточно информативен для принятия решения об окончании испытаний, поэтому возникает необходимость выбора показателей, характеризующих достаточность объема проведенных испытаний для проверки корректного выполнения функций, возможно, на основе совместной оценки принятых при исследовании показателей.

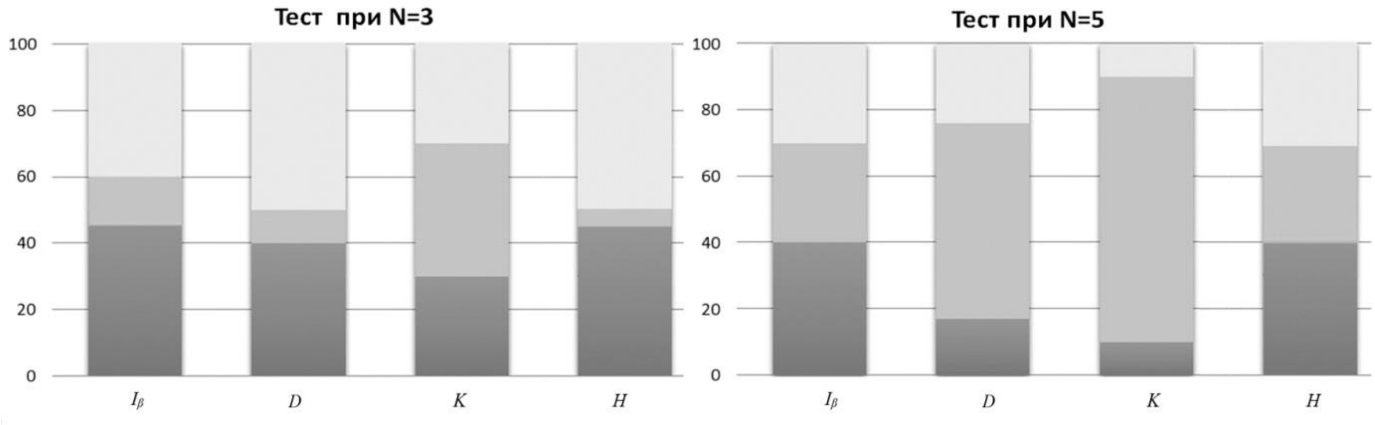



Рисунок 4.8 – Результаты тестирования программного обеспечения устройства

Анализ преимуществ предлагаемого подхода показал, что использование предложенной методики не менее чем на 5% сокращает объем испытаний для трех функций (тестирование устройства в режиме приема данных) и не менее чем на 10% – для пяти функций (тестирование устройства в режиме обновления данных) по сравнению с классическим тестированием.

Для имитации неисправностей испытания в процессе тестирования было определено три режима, которые требовали повышения устойчивости к отказам и сбоям в результате классического тестирования:

– режим передачи информации в режимах работы устройства (P1);

– режим обмена информацией с другими устройствами комплекса (Р2);

– режим настройки и диагностики изделия с рабочей станции (Р3).

В связи с полученной проблемой детектирования и последующего устранения ошибок программного обеспечения проведена интенсивная имитация с применением показателей, полученных с помощью критерия окончания испытаний. Приведены данные исследования микропроцессорного устройства с помощью алгоритма определения критерия окончания испытаний. Для режимов работы устройства определены значения показателей, которые характеризуют набор необходимого объема для достижения работоспособности.

Таблица 4.10 – Результаты эксперимента

| № | Тип ошибки | Признаки проявления | Обнаружено проявлений с помощью имитации неисправностей, шт. | Дополнительно обнаруженные ошибки, шт. | Дополнительно обнаруженные ошибки, % |
|---|---|---|---|---|---|
| 1 | Ошибки типов | Некорректные результаты вычислений в режимах | 10 | 0 | 0 |
| 2 | Ошибки переходов программы | Искажение алгоритма просмотра графического интерфейса | 45 | 44 | 98 |
| 3 | Ошибки управления ресурсами | Нарушение передачи/нарушение пользовательских операций | 84 | 77 | 91 |
| 4 | Ошибки аппаратных модулей | Потеря управления мощностью передатчика | 23 | 20 | 86 |
| 5 | Ошибки интерфейса | Потеря принятой информации; искажение данных | 17 | 7 | 41 |
| Общее число ошибок | | | 179 | 148 | 83 |

На рисунке 4.9 приведены результаты сравнения времени поиска неисправностей с помощью стандартных средств и при использовании испытательного стенда с имитацией неисправностей программными средствами.

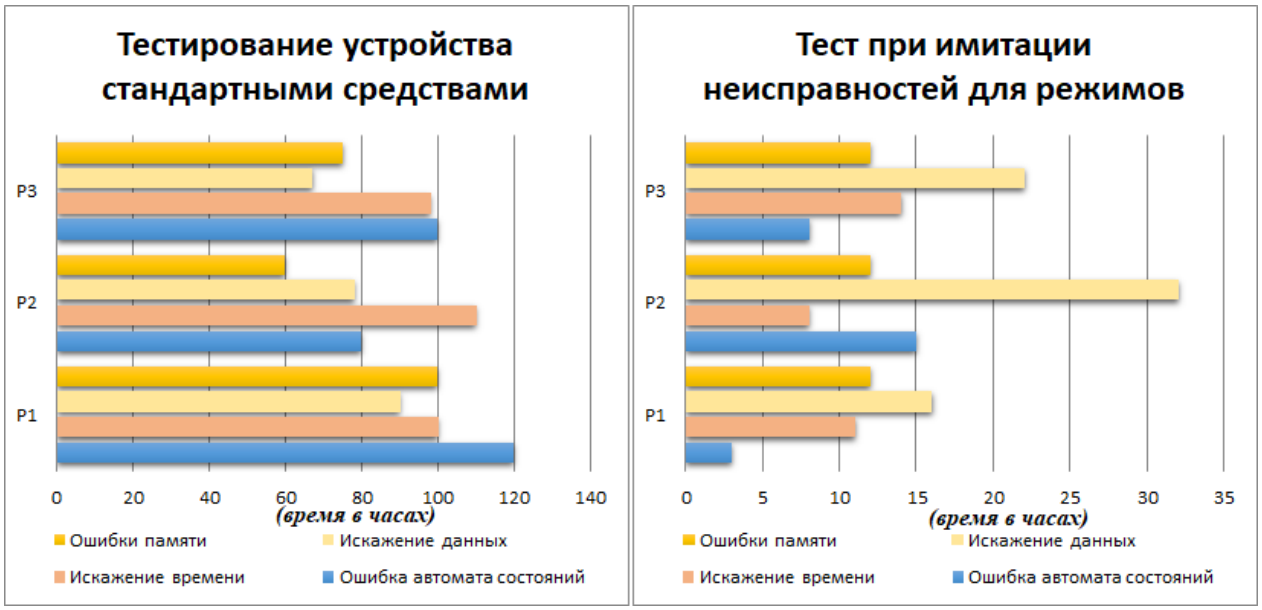



Рисунок 4.9 – Поиск ошибок программного обеспечения для режимов P1-P3

Следует отметить, что для всех режимов испытаний поиск ошибок программного обеспечения занял значительно меньше времени. Максимальное время поиска для имитации неисправностей – *32* часа, а для ручного поиска – *120* часов. Результаты поиска ошибок с помощью стенда демонстрируют, что по сравнению с ручным поиском количество обнаружения значительно выше (на 82% от общего числа обнаруженных ошибок).

## 4.4 Выводы по главе

1. Получены результаты испытаний на разработанном макете имитатора неисправностей для проверки его функций по внесению искажений во входные данные и обнаружению отказов и сбоев с помощью анализа реакции микропроцессорного устройства на инъектируемые ошибки. Установлено, что независимое применение имитатора позволяет обеспечить повышение количества найденных ошибок, способных приводить к отказам и сбоям (от *5*% до *19*% выявленных ошибок от числа выявленных с помощью методов классического тестирования программного обеспечения).

2. С помощью разработанных средств программного комплекса имитации системных событий и автоматизации осуществления испытаний выполнены работы по определению основных функций программного обеспечения и статистики распределения времени использования функций в режимах работы микропроцессорного устройства.

3.Установлено, что имитация неисправностей с направленным фаззингом эффективнее по сравнению с использованием классического тестирования (*179* случаев обнаружения проявлений ошибок ПО вместо *31*). Предложенные алгоритмы имитации неисправностей и анализа реакции также позволили выявить ошибки программного обеспечения для прототипа быстрее, чем с помощью применения классических методов тестирования (*32* часа вместо *120*).

4. Полученные значения показателей для положительного критерия окончания испытаний с помощью методики на основе нечеткого логического вывода, характеризующего необходимый объем тестовых испытаний микропроцессорного устройства, соответствуют состоянию работоспособности микропроцессорного устройства.

# ЗАКЛЮЧЕНИЕ

В результате проведенных исследований получены новые научные и практические результаты, направленные на повышение устойчивости к отказам и сбоям микропроцессорных устройств за счет обнаружения ошибок ПО, проявляющихся в реальной аппаратной среде.

1. В результате анализа работ, посвященных вопросам устойчивости программно-технических комплексов к отказам и сбоям, выявлено, что известные методики обнаружения неисправностей на основе регламентируемых методик испытаний устройств недостаточно эффективны и требуют усовершенствования. В связи с тем, что используемые микропроцессорные элементы аппаратных средств относятся к классу высоконадёжных изделий, то основной проблемой обеспечения работоспособности проектируемых устройств является выявление отказов, связанных с дефектами программного обеспечения, проявляемыми в реальной аппаратной среде, для чего требуется имитация неисправностей.

2. Исследованы методы автоматизации искажения данных и анализа реакции на внесенные ошибки в проектируемую систему. Рассмотрены различия между результатами тестирования на основе внесения неисправностей в функциональные компоненты устройства и на основе имитации последствий возникновения неисправностей. Показано, что использование второго метода имеет преимущества для микропроцессорных систем в связи с отсутствием разрушающих действий и уменьшением временных затрат на его осуществление.

3. Использование модифицированного алгоритма фаззинга, включающего предобработку исходных данных с определением режимов испытания устройства, проверяемых функций и точек контроля, позволяет быстрее выявить ошибки с помощью фаззинга для графа ветвлений программы микропроцессорного устройства и анализа реакции устройства с помощью интерфейса JTAG на проимитированные неисправности программы (с помощью изучения генерируемой матрицы ветвлений).

4. Полученные значения показателей для положительного решения об завершении испытаний на основе критерия, значения которого с помощью методики на основе нечеткого логического вывода, характеризующего необходимый объем тестовых испытаний микропроцессорного устройства, соответствует состоянию работоспособности микропроцессорного устройства.

5. Выявлено, что независимое применение имитатора позволяет повысить количества найденных ошибок, способных приводить к отказам и сбоям (от *5*% до *19*% выявленных по сравнению с классическим тестированием), а совместное применение имитатора с направленным фаззингом позволяет обеспечить работоспособность устройства.

6. Разработанный программно-аппаратный комплекс, основанный на совместном применении предлагаемой методики анализа проблемных ситуаций, модифицированного алгоритма фаззинга, а также имитатора неисправностей, позволяет практически осуществить автоматизацию проведения испытаний микропроцессорных устройств, и сократить временные затраты (*32* часа вместо *120*), а также увеличить количество выявленных дефектов ПО по сравнению с классическим тестированием на 82% от общего числа ошибок (с 31 до 179 ошибок).

# СПИСОК СОКРАЩЕНИЙ

АРМ – автоматизированное рабочее место;

АСУ – автоматизированная система управления;

МК – микроконтроллер;

ОЗУ – оперативное запоминающее устройство;

ОИ – объект испытаний;

ПЛИС – программируемая логическая интегральная схема;

ПО – программное обеспечение;

РЭА – радиоэлектронная аппаратура;

САПР – система автоматизированного проектирования;

СБИС – сверхбольшие интегральные схемы

СУ – станция управления;

EN 50128 – европейский стандарт для обеспечения безопасности «Железнодорожные приложения - Системы связи, сигнализации и обработки»;

ARM – семейство готовых топологий микропроцессорных/микроконтроллерных ядер на основе усовершенствованной RISC-машины;

AVR – семейство RISC-микроконтроллеров;

DSP – цифровой сигнальный процессор;

FI – внесение неисправностей;

FMEA – анализ видов и последствий отказов;

FMECA – анализ критичности отказов;

FMEDA –диагностический анализ последствий отказов;

FPGA – программируемая логическая интегральная схема;

IEC 61508 – международный стандарт «Функциональная безопасность электрических / электронных / программируемых электронных систем, связанных с безопасностью»;

JTAG – стандарт тестовой архитектура граничного сканирования IEEE 1149.1;

MIPS – система команд и микропроцессорных архитектур, разработанных компанией MIPS.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Авиженис, А. Отказоустойчивость — свойство, обеспечивающее постоянную работоспособность цифровых систем / А. Авиженис // Труды института инженеров по электронике и радиоэлектронике. – 1978. – Т. 66, № 10. – С. 5–25.

2. Автоматизация синтеза и обучение интеллектуальных систем автоматического управления: [сборник] / Рос. акад. наук, Отд-ние информ. технологий и вычислительных систем; отв. ред.: И. М. Макаров, В. М. Лохин. – Москва: Наука, 2009. – 227, [1] с.: ил., табл.

3. Балашов, Е. П. Проектирование информационно-управляющих систем / Е. П. Балашов, Д. В. Пузанков. – Москва: Радио и связь, 1987. – 256 с.

4. Брюкин, В. Н. Анализ надежности микроэлектронных систем при автоматизированном проектировании / В. Н. Брюкин, М. Х. Булатов – М.: Радио и связь, 1984. – 216 с.

5. Василенко, Н. Н. Модели оценки надежности программного обеспечения / Н. Н. Василенко, В. А. Макаров // Вестник новгородского государственного университета. – 2004. – № 28. – С. 126–132.

6. Вентцель, Е. С. Теория вероятностей / Е. С. Вентцель. – Москва: Наука, 1969. – 576 с.

7. Гобл, В. М. FMEA: анализ видов и последствий отказов / В. М. Гобл. – Текст: электронный // Электронная библиотека «UA Automation»: [сайт]. – URL: http://ua.automation.com/content/fmea-analiz-vidov-i-posledstvij-otkazov (дата обращения: 1.03.2021).

8. ГОСТ 15467–79. Управление качеством продукции. Основные понятия. Термины и определения: межгос. стандарт: изд. офиц.: утв. и введ. в действие Постановлением Государственного комитета СССР по стандартам от 26 янв. 1979 г. № 244: дата введ. 1979.07.01. – Москва: Стандартинформ, 2009. – 21 с.

9. ГОСТ 27.002–2015. Надежность в технике. Термины и определения: гос. стандарт Рос. Федерации: изд. офиц.: утв. и введ. в действие Постановлением

Госстандарта России от 10 июня 2014 г. № 519–ст.: введ. впервые: дата введ. 2017.03.01. – Москва: Стандартинформ, 2016. – 28 с.

10. ГОСТ Р 51904–2002. Программное обеспечение встроенных систем. Общие требования к разработке и документированию: гос. стандарт Рос. Федерации: изд. офиц.: утв. и введ. в действие Постановлением Госстандарта России от 25 июня 2002 г. № 247–ст.: введ. впервые: дата введ. 2003.07.01. – Москва: Стандартинформ, 2005. – 62 с.

11. ГОСТ Р ИСО 26262–9–2014. Национальный стандарт Российской Федерации. Дорожные транспортные средства функциональная безопасность: гос. стандарт Рос. Федерации: изд. офиц.: утв. и введ. в действие Постановлением Госстандарта России от 18 мая 2015 г. № 364–ст: введ. впервые: дата введ. 2015.05.01. – Москва: Стандартинформ, 2015. – 23 с.

12. ГОСТ Р МЭК 61508–7–2012. Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью: гос. стандарт Рос. Федерации: изд. офиц.: утв. и введ. в действие Постановлением Госстандарта России от 29 октября 2012 г. № 592–ст.: введ. впервые: дата введ. 2013.08.01. – Москва: Стандартинформ, 2014. – 100 с.

13. ГОСТ Р МЭК 62061-2015. Безопасность оборудования. Функциональная безопасность систем управления электрических, электронных и программируемых электронных, связанных с безопасностью: гос. стандарт Рос. Федерации: изд. офиц.: утв. и введ. в действие Постановлением Госстандарта России от 10 июня 2014 г. № 519–ст.: введ. впервые: дата введ. 2015.02.25. – Москва: Стандартинформ, 2016. – 78 с.

14. ГОСТ РВ 27.1.02–2005. Надежность военной техники. Программа обеспечения надежности. Общие требования: Гос. воен. стандарт Рос. Федерации: утв. и введ. Федер. агентством по техн. регулированию и метрологии: введ. впервые: дата введ. 2006.01.01. – Москва: Стандартинформ, 2005. – III, 14 с.

15. Денисова, Л. А. Многокритериальная оптимизация на основе генетических алгоритмов при синтезе систем управления: монография / Л. А. Денисова; Ом. гос. техн. ун-т. – Омск: Изд-во ОмГТУ, 2014. – 170 с.

16. Денисова, Л. А. Модели и методы проектирования систем управления объектами с переменными параметрами: монография / Л. А. Денисова; Ом. гос. техн. ун-т. – Омск: Изд-во ОмГТУ, 2014. – 167 с.

17. Дьяконов, В. П. MATLAB 7. */R2006/R2007: самоучитель / В. П. Дьяконов. – Москва: ДМК Пресс, 2008. – 768 с.

18. Ермаков, А. Д. К синтезу адаптивных проверяющих последовательностей для недетерминированных автоматов / А. Д. Ермаков, Н. В. Евтушенко // Труды Института системного программирования РАН. – 2016. – Т. 28, вып. 3. – С. 123–144.

19. Ермаков, М. К. Проведение динамического анализа исполняемого кода формата ARM ELF на основе статического бинарного инструментирования / М. К. Ермаков // Научно-технические ведомости Санкт-Петербургского государственного политехнического университета. Информатика. Телекоммуникации. Управление. – 2016. – № 1 (236). – С. 108–117.

20. Ермилов, В. А. Метод отбора существенных неисправностей для диагностики цифровых схем / В. А. Ермилов // Автоматика и телемеханика. – 1971. – № 1. – С. 159– 167.

21. Иванов, В. И. Исследование протоколов маршрутизации в негеостационарных спутниковых системах / В. И. Иванов // T-Comm − Телекоммуникации и Транспорт. – 2012. – №6. – С. 19−21.

22. Ильин, С. Фаззинг, фаззить, фаззер: ищем уязвимости в программах, сетевых сервисах, драйверах / С. Ильин // Хакер. – 2010. – № 7. – С. 32–36.

23. Иныксон, В. М. Исследование систем автоматизации обнаружения дефектов в исходном коде программ / В. М. Иныксон, М. Ю. Моисеев, В. А. Цесько // Научно-технические ведомости Санкт-Петербургского государственного политехнического университета. Информатика. Телекоммуникации. Управление. – 2008. – № 5. – С. 119–124.

24. Исследование систем Ст тр Ст* и C.Vmp1. Опыт обеспечения отказоустойчивости в мультипроцессорных системах / Д. П. Северек, В. Кини, Х.

Мэшберн и [др.] // Труды института инженеров по электронике и радиоэлектронике. – 1978. – Т. 66, № 10. – С. 89–117.

25. Ицкович, Э. Л. Эволюция средств и систем автоматизации технологических процессов / Э. Л. Ицкович // Автоматизация в промышленности. – 2009. – № 8. – С. 3–10.

26. Иыуду, К. А. Прогнозирование надежности программ на ранних этапах разработки / К. А. Иыуду, А. И. Касаткин, В. В. Бахтизии // Надежность и контроль качества. – 1982. – № 5. – С. 18–30.

27. Патент № 2392657 Российская Федерация, МПК G06F11/22 (2006.01). Автоматизированное устройство для тестирования микропроцессорных систем № 2008103655/09: заявл. 30.01.2008: опубл. 20.06.2010 / Д. И. Дейнека, В. В. Лучинин. – 7 с.

28. Катсуки, Д. Piuribus – отказоустойчивый операционный мультипроцессор / Д. Катсуки, Э. С. Элсам, У. Д. Манн // Труды института инженеров по электронике и радиоэлектронике. – 1978. – Т. 66, № 10. – С. 49–68.

29. Ковалев, И. В. Анализ проблем в области исследования надежности программного обеспечения: многоэтапность и архитектурный аспект / И. В. Ковалев // Сибирский журнал науки и технологий. – 2014. – № 3. – С. 78–92.

30. Копылов, С. А. Обзор методов и подходов обнаружения отказов и отказоустойчивости системы управления для роботизированных морских подвижных объектов / С. А. Копылов // Инженерный вестник Дона. – 2014. – № 3. – С. 16–20.

31. Крузе, Н. РАД: Сопроцессор серии 6800 для обнаружения неисправностей в микрокомпьютерах / Н. Крузе, Ж. Шавад // Труды института инженеров по электронике и радиоэлектронике. – 1986. – Т. 74, № 5. – С. 119–127.

32. Ксенз, С. П. Диагностика и ремонтопригодность радиоэлектронных средств / С. П. Ксенз. – М.: Радио и связь. – 1989. – 248 с.

33. Легков, К. Е. Новые принципы построения автоматизированных систем управления современными инфокоммуникационными сетями специального назначения / К. Е. Легков // Наукоемкие технологии в космических исследованиях Земли. – 2015. – № 1. – С. 38–41.

34. Липаев, В. В. Проблемы обеспечения надежности и устойчивости сложных комплексов программ АСУ / В. В. Липаев // Управляющие системы и машины. – 1977. – № 3. – С. 39–45.

35. Львович, И. Я. Модель выбора вариантов резервирования в системе управления стендовыми испытаниями / И. Я. Львович, А. А. Тузиков // Вестник Воронежского государственного технического университета. – 2011. – № 3. – С. 47–50.

36. Макаров, А. Н. Метод автоматизированного поиска программных ошибок в алгоритмах обработки сложноструктурированных данных / А. Н. Макаров // Прикладная дискретная математика. – 2009. – №3. – С. 117–127.

37. Макконнелл, С. Совершенный код. Мастер класс / С. Макконнелл. – 2-е изд. – Москва: Русская редакция, 2010. – 896 с.

38. Марков, И. Автоматическое тестовое оборудование с подвижными пробниками в производстве электронных изделий / И. Марков, И. Рыков // Компоненты и технологии. – 2005. – № 1. – С. 168–170.

39. Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. – Санкт-Петербург: Питер, 2018. – 351 с. – (Библиотека программиста).

40. Новиков, Д. В. К вопросу о выборе среды для имитационного моделирования систем железнодорожной автоматики и телемеханики / Д. В. Новиков // Актуальные вопросы развития систем железнодорожной автоматики и телемеханики. – 2013. – № 1. – С. 60–65.

41. Ногин, В. Д. Проблемы сужения множества Парето: подходы к решению / В. Д. Ногин // Искусственный интеллект и принятие решений. – 2008. – № 1. – С. 98–112.

42. Основы технической диагностики: в 2 кн. Кн. 1. Модели объектов, методы и алгоритмы диагноза / В. А. Карибский, П. П. Пархоменко, Е. С. Согомонян и др.; под ред. П. П. Пархоменко. – Москва: Энергия, 1976. – 462 с.

43. Панков Д. А. Контроль и диагностика неисправностей программно-аппаратного комплекса / Д. А. Панков, Л. А. Денисова // Омский научный вестник. – 2018. – № 2 (158). – С. 128–133.

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

45. Панков Д. А. Разработка и исследование алгоритма маршрутизации в многофункциональном комплексе связи. / Д. А. Панков, Л. А. Денисова // Омский научный вестник. – 2017. – № 6 (156). – С. 143–145.

46. Патент № 2549523 Российская Федерация, МПК G06F 11/00 (2006.01), G05B 23/00(2006.01). Способ мутационного тестирования радиоэлектронной аппаратуры и ее управляющего программного обеспечения № 2014117399/08: заявл. 29.04.2014: опубл. 27.04.2015 / Д. А. Недорезов. – 7 с.

47. Патент № 2696977 Российская Федерация, МПК H04B1/40(2015-01-01). Носимая автоматизированная радиостанция диапазона КВ-УКВ: № 2018141019: заявл. 21.11.2018: опубл. 08.08.2019 / В. В. Фомин, А. В. Мартынов, А. В. Лушпай, А. В. Черненко, П. А. Басов, Д. А. Панков. – 13 с.

48. Патент № 2697629 Российская Федерация, МПК G06F 11/261 (2018.08). Устройство для имитации неисправностей в программно-аппаратных системах № 2018105476 заявл. 13.02.18; опубл.15.08.19 / Д. А. Панков. – 8 с.

49. Печенкин, А. И. Архитектура масштабируемой системы фаззинга сетевых протоколов на многопроцессорном кластере / А. И. Печенкин, А. В. Никольский // Проблемы информационной безопасности. Компьютерные системы. – 2013. – № 2. – С. 73–80.

50. Пинкевич, В. Ю. Тестирование и отладка встраиваемых вычислительных систем на основе уровневых моделей / В. Ю. Пинкевич, А. Е. Платунов // Научно-технический вестник информационных технологий, механики и оптики. – 2018. – Т. 18, № 5. – С. 801–808.

51. Полонников, Р. И. Методы оценки надежности программного обеспечения / Р. И. Полонников, А. В. Никандров. – Санкт-Петербург: Политехника, 1992. – 80 с.

52. Похабов, Ю. В. Надежность в цифровых технологиях / Ю. В. Похабов // Надежность. – 2020. – Т. 20, № 2. – С. 3–11.

53. Протоколы геомаршрутизации самоорганизующихся мобильных сетей / Д. Е. Прозоров, А. П. Метелев, А. В. Чистяков [и др.] // T-Comm –Телекоммуникации и Транспорт. – 2012. – № 5. – С. 16–19.

54. Реннелс, Д. А. Архитектура космических бортовых вычислительных систем, устойчивых к отказам / Д. А. Реннелс // Труды института инженеров по электронике и радиоэлектронике. – 1978. – Т. 66, № 10. – С. 186–206.

55. Розенталь, Р. М. Мировой и российский опыт развития FMEA / Р. М. Розенталь // Стандарты и качество. – 2010. – № 4. – С. 54–56.

56. Рутковская, Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский. – Москва: Горячая линия. – Телеком, 2002. – 452 с.

57. Савкин, Л. В. Мажоритирование логико-арифметических операций на низком аппаратном уровне в реконфигурируемой диагностической модели бортового комплекса управления космического аппарата / Л. В. Савкин // Universum: технические науки. – 2014. – № 6. – С. 143–145.

58. Сандерс, Д. Искусственный интеллект в сенсорных системах / Д. Сандерс (D.Sanders) // Control engineering. Россия. – 2014. – № 1 (49). – С. 35–39.

59. Свидетельство о государственной регистрации программы для ЭВМ № 2019661730 Российская Федерация. Программа автоматизированного тестирования для поиска неисправностей портативного радиоприемного устройства: *№* 2019660514; заяв. 27.08.2019; опубл. 05.09.2019. / Панков Д. А., Кондратьев А. Н.; заявитель и патентообладатель АО «ОНИИП».

60. Свидетельство о государственной регистрации программы для ЭВМ № 2019664742 Российская Федерация. Специальное программное обеспечение тестирования и имитации для приемного и передающего трактов радиостанции:

№ 2019663695; заяв. 01.11.2019; опубл. 13.11.2019 / Панков Д. А., Козьма П. А.; заявитель и патентообладатель АО «ОНИИП».

61. Селлерс, Ф. Методы обнаружения ошибок в работе ЭЦВМ / Ф. Селлерс. – Москва: Мир, 1972. – 310 с.

62. Скляр, В. В. Сертификация информационно-управляющей платформы на базе ПЛИС на соответствие требованиям по функциональной безопасности стандарта МЭК 61508 / В. В. Скляр – Текст: электронный // Электронная библиотека «Nuclear Journal»: [сайт]. – 2020. – URL: https://www.nuclear-journal.com/index.php/journal/article/download/459/375/ (дата обращения: 11.04.2019).

63. Слинкин, Д. И. Анализ современных методов тестирования и верификации проектов сверхбольших интегральных схем / Д. И. Слинкин // Программные продукты и системы. – 2017. – Т. 30, № 3. – С. 401 – 406.

64. Строганов, А. Обзор программных комплексов по расчету надежности сложных технических систем / А. Строганов, В. Жданов, С. Полесский. – Текст: электронный // Электронная библиотека «Cyberleninka»: [сайт]. – 2020. – URL: https://cyberleninka.ru/article/n/obzor-programmnyh-kompleksov-po-raschetu-nadezhnosti-slozhnyh-tehnicheskih-sistem (дата обращения: 01.03.2021).

65. Полесский, С. Разработка методики выявления факторов, наиболее сильно влияющих на надежность электронных устройств / С. Полесский, О. Маякова, А. Алейников // Системы управления, связи и безопасности. – 2016. – № 4. – С. 114–127.

66. Тейер, Т. Надежность программного обеспечения / Т. Тейер, М. Липов, Э. Нельсон. – Москва: Мир, 1981. – 324 с.

67. Технические средства диагностирования: справочник / В. В.Клюев, П. П. Пархоменко, В. Е. Абрамчук [и др.]; под общ. ред. В. В. Клюева. – М.: Машиностроение, 1989. – 672 с.

68. Той, В. Н. Проектирование отказоустойчивых местных процессоров для систем электронной коммутации / В. Н. Той // Труды института инженеров по электронике и радиоэлектронике. – 1978. – Т. 66, № 10. – С. 26–48.

69. Томилов, И. О. Фаззинг. Поиск уязвимостей в программном обеспечении без наличия исходного кода / И. О. Томилов, А. В. Трифанов // Интерэкспо Гео-Сибирь. – 2017. – Т. 9, № 2. – С. 75–80.

70. Увалов, Д. В. Бортовая микропроцессорная система управления с повышенной сбоеустойчивостью / Д. В. Увалов // Решетневские чтения: материалы XVIII Междунар. науч. конф., посвящ. 90-летию со дня рождения генер. конструктора ракетно-космич. систем акад. М. Ф. Решетнева (Красноярск, 11–14 нояб. 2014 г.): в 3 ч. / под общ. ред. Ю. Ю. Логинова; Сиб. гос. аэрокосмич. ун-т. – Красноярск: ИП Михайлова И. Г., 2014. – Ч. 1. – С. 248–250.

71. Ушаков, И. А. Надёжность: прошлое, настоящее, будущее / И. А. Ушаков – Текст: электронный // Электронная библиотека «Cyberleninka»: [сайт]. – URL: https://cyberleninka.ru/article/n/nadyozhnost-proshloe-nastoyaschee-buduschee (дата обращения: 21.03.2018).

72. Чекмарев, С. А. Проектирование системы инъекции ошибок для отработки сбоеустойчивых процессоров бортовых систем малого космического аппарата / С. А. Чекмарев, В. Х. Ханов // Решетневские чтения: материалы XVIII Междунар. науч. конф., посвящ. 90-летию со дня рождения генер. конструктора ракетно-космич. систем акад. М. Ф. Решетнева (Красноярск, 11–14 нояб. 2014 г.): в 3 ч. / под общ. ред. Ю. Ю. Логинова; Сиб. гос. аэрокосмич. ун-т. – Красноярск: ИП Михайлова И. Г., 2014. – Ч. 1. – С. 254–255.

73. Чекмарев, С. А. Технология инъектирования сбоев для тестирования сбоеустойчивости микропроцессоров, предназначенных к использованию в бортовой аппаратуре космических аппаратов / С. А. Чекмарев, В. Х. Ханов, А. С. Тимохович // Вестник СибГАУ. – 2016. – Том 17, № 3. – С. 768–781.

74. Черкесов, Г. Н. Надежность аппаратно-программных комплексов: учеб. пособие / Г. Н. Черкесов. – Санкт-Петербург: Питер, 2005. – 479 с.

75. Черноруцкий, И. Г. Методы оптимизации в теории управления / И. Г. Черноруцкий. – Санкт-Петербург: Питер, 2004. – 256 с.

76. A Fault Injection Tool for Distributed Heterogeneous Embedded Systems / A. Dasilva, J.-F. Martínez, L. López-Santidrián [et al.] // Telematics and Inforamtion

Systems: proceedings of the Euro American Conference (May 2007). – 2007. – P. 17–23.

77. A Fault-Tolerant Embedded Microcontroller Testbed / D. Rennels, D. Caldwell, R. Hwang, M. Mesarina. – DOI: 10.1109/PRFTS.1997.640118 // Pacific Rim International Symposium on Fault-Tolerant Systems: proceedings (Taipei, Taiwan, 15–16 Dec. 1997). – Taipei, 1997. –*URL:* https://www.researchgate.net/publication/3724046_A_fault-tolerant_embedded_microcontroller_testbed (date accessed: 20.01.2017).

78. A systematic review of fuzzing based on machine learning techniques / Y. Wang, P. Jia, L. Liu, J. Liu. – DOI: 10.1371/journal.pone.0237749. – URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0237749 (date accessed: 11.09.2018).

79. Aerospace Standard ARP5580. Recommended Failure Modes and Effects Analysis (FMEA) for non-Automobile Applications: issued 07.2001: reaffirmed 08.2020 / SAE International in United States. – URL: https://saemobilus.sae.org/content/ARP5580/ *(*date accessed: 01.12.2019*).*

80. Antoni, L. Using Run-Time Reconfiguration for Fault Injection Applications / L. Antoni, R. Leveugle, B. Feher. – DOI: 10.1109/TIM.2003.817144 // IEEE Transactions on Instrumentation and Measurement. – 2003. – Vol. 52, № 5. – P. 1468–1473.

81. Arlat, J. Fault Injection for Dependability Validation: Methodology and Some Application / J. Arlat, M. Aguera, L. Amat, Y.Crouzet, J.-Ch. Fabre, J.-C. Laprie, E. Martins, D. Powell // A IEEE Transactions on Software Engineering. – Vol. 16, № 2. – 1990. – P. 166 – 182.

82. Avizienis, A. Basic Concepts and Taxonomy of Dependable and Secure Computing / A. Avizienis, J. C. Laprie // IEEE Transactions on Dependable and Secure Computing. – 2004. – Vol. 1. – P. 11–33.

83. Avizienis, A. Design of fault-tolerant computers / A. Avizienis. – DOI:10.1145/1465611.1465708 // AFIPS'67 (Fall) fall joint computer conference: proceedings (14–16 November 1967). – Anaheim, 1967. – P. 733–743. – *URL:*

https://www.researchgate.net/publication/24149601_Design_of_faulttolerant_computers (date accessed: 10.01.2019).

84. Avizienis, A. Systematic Design of Fault-Tolerant / A. Avizienis // Safe Comp 96: the 15th International Conference on Computer Safety, Reliability and Security (Vienna, Austria 23–25 October 1996). – London: Springer, 1997. – P. 3–18.

85. Barton, J. Fault Injection Experiments Using Fiat / J. H. Barton, E. W. Czeck, Z. Z. Segall, D. P. Siewiorek // EEE Transactions on Computers − 1990. − Vol. 39. − P. 575–582.

86. Carreira, J. V. Fault injection spot-checks computer system dependability / J. V. Carreira, D. Costa, J. G. Silva. – DOI: 10.1109/6.780999 // IEEE Spectrum. – 1999. – Vol. 36, № 8. – P. 50–55.

87. Chen, L. N-version programming: A fault-tolerance approach to reliability of software operation / L. Chen, A. Avizienis // FTCS-8 – 1978.

88. Combining dynamic symbolic execution, code static analysis and fuzzing / A. Y. Gerasimov, S. S. Sargsyan, S. F. Kurmangaleev [et al.]. – DOI: 10.15514/ISPRAS-2018-30(6)-2 // Proceedings of the Institute for System Programming of the RAS (Proceedings of ISP RAS). – 2018. – Vol. 30 (6). – P. 25–38. – URL: https://cyberleninka.ru/article/n/combining-dynamic-symbolic-execution-code-static-analysis-and-fuzzing (date accessed: 20.03.2017).

89. Diegues, N. Self-Tuning Intel Transactional Synchronization Extensions / N. Diegues, P. Romano // 11th International Conference on Autonomic Computing : proceedings (Philadelphia, PA, 18–20 June 2014). – Philadelphia, 2014. – P. 209–219. – URL: https://doi.org/10.1145/1352694.1352712 (date accessed: 11.05.2019).

90. Failure Modes and Effects Analysis for a Software-Intensive Satellite Control System / M. Hecht, E. Shokri, S. Meyers [et al.] // 2008 International Systems Safety Conference: proceedings (Vancouver, August 2008). – Vancouver, 2008. – URL: https://www.researchgate.net/publication/265851785_Failure_Modes_and_Effects_Analysis_for_a_Software-Intensive_Satellite_Control_System (date accessed: 17.07.2020).

91. Failure Resilience for Device Drivers / J. N. Herder, H. Bos, B. Gras [et al.] // Dependable Systems and Networks (DSN'07): proceedings of the 37th Annual IEEE/IFIP International Conference (Edinburgh, 25–28 June 2007). – Edinburgh, 2007. – P. 41–50.

92. Fault Injection in the Automotive Standard ISO 26262: An Initial Approach / L. Pintard, J. Fabre, K. Kanoun [et al.] // Dependable Computing: 14th European Workshop, EWDC 2013: proceedings (Coimbra, Portugal, 15–16 May 2013). Vol. 7869: Lecture Notes in Computer Science. – Berlin; Heidelberg: Springer, 2013. – P. 126–133. – *URL:* https://link.springer.com/book/10.1007%2F978-3-642-38789-0 (date accessed: 17.09.2018).

93. GREYONE: Data Flow Sensitive Fuzzing / S. Gan, C. Zhang, P. Chen [et al.]. – URL: https://www.usenix.org/system/files/sec20spring_gan_prepub.pdf (date accessed: 17.06.2019).

94. Han, H. IMF: Inferred Model-based Fuzzer / H. Han, S. Cha. – DOI:10.1145/3133956.3134103 // Computer and Communications Security: proceedings of the 2017 ACM SIGSAC conference (Dallas, Texas USA, October2017). – New York: Association for Computing Machinery, 2017. – P. 2345–2358.

95. Herder, J. Failure Resilience for Device Drivers / J. Herder, H. Bos, B. Gras // Proceeding DSN '07 Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. – 2007. – P. 41–50.

96. Hsueh, M. Fault Injection Techniques and Tools / M.-C. Hsueh, T. K. Tsai, R. K. Iyer. – DOI: 10.1109/2.585157 // Computer. – 1997. – Vol. 30 (4). – P. 75–82.

97. IEEE Std 1149.1–2001. IEEE Standard Test Access Port and Bondary-Scan Architecture: approved 14.06.2001: published 23.07.2001. –2001. – URL: https://fiona.dmcs.pl/~cmaj/JTAG/JTAG_IEEE-Std-1149.1-2001.pdf (date accessed: 17.06.2019).

98. Kennon, R. PMEA Technique for Microcomputer Assemblies / R. Kennon, R. Newell // Annual Reliability and Maintainability Symposium: proceedings (Los Angeles, California, Gan., 1982). – New York, 1982. – P. 78–83.

99. Lefebvre, M. Functional test and diagnosis: a proposed JTAG sample mode scan tester / M. Lefebvre. – DOI: 10.1109/TEST.1990.114035 // International Test Conference: proceedings (Washington, DC, USA, 10–14 Sept. 1990). – Washington, 1990. – URL: https://ieeexplore.ieee.org/document/114035 (date accessed: 12.12.2019).

100. Legg, J. M. Computerized Approach for Matrix-Form FMEA / J. M. Legg. – DOI: 10.1109/TR.1978.5220355 // IEEE Transactions on Reliability. – 1978. – Vol. R-27 (4). – P. 254–257.

101. Liestman, A. Fault-Tolerant Scheduling Problem. IEEE Trans. on Software Engineering. – 1986. – Vol. 12. – P. 1089–1095.

102. Lyu, M. Assuring design diversity in N-version software: a design paradigm for N-version programming / M. Lyu, A. Avizienis // Dependable Computing for Critical Applications 2. Vol. 6: Dependable Computing and Fault-Tolerant Systems. – Wien; New York: Springer-Verlag, 1992. – P. 197–218. – URL: https://link.springer.com/content/pdf/10.1007%2F978-3-7091-9198-9.pdf (date accessed: 08.03.2019).

103. Medoff, M. Functional Safety – An IEC 61508 SIL 3 Compatible Development Process – exida.com L. L. C. / M. Medoff, R. Faller // Sellersville, PA, USA. – 2010. – 281 p.

104. Miller, B. Anywhere, Any Time Binary Instrumentation / B. Miller, A. Bernat // ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. – 2011. – P. 9–16.

105. Neuzz: Efficient fuzzing with neural program smoothing / D. She, K. Pei, D. Epstein [et al.] // IEEE Symposium on Security and Privacy (San Francisco, CA, 20–22 May 2019). – San Francisco, 2019. – P. 803–817. https://deepai.org/publication/neuzz-efficient-fuzzing-with-neural-program-learning (date accessed: 08.03.2019).

106. Optimizing seed selection for fuzzing / Alexandre Rebert, Sang Kil Cha, Thanassis Avgerinos [et al.] // IEEE Access. – 2018. – Vol. 6. – P. 861– 875. – URL: https://www.usenix.org/conference/usenixsecurity14/technicalsessions/presentation/re

bert (date accessed: 11.04.2019).

107. Pankov D. A. Automated testing and fault diagnosis of the microcontroller system / D. A. Pankov, L. A. Denisova // IOP Conference Series: Materials Science and Engineering. International Workshop "Advanced Technologies in Material Science, Mechanical and Automation Engineering – MIP: Engineering – 2019". Krasnoyarsk Science and Technology City Hall of the Russian Union of Scientific and Engineering Associations. – 2019. – P. 22072.

108. Pankov, D. A. Model studies of systems with diagnostics based on fault simulation / D. A. Pankov, L. A. Denisova // IOP Conference Series: Materials Science and Engineering. Krasnoyarsk Science and Technology City Hall of the Russian Union of Scientific and Engineering Associations. – 2020. – P. 12021.

109. Patent № 7908520B2, US, IPC G06F11/2028. Self-testing and-repairing fault-tolerance infrastructure for computer systems : application 06.20.2001 : publ. 15. 03.2011 / A. Avizienis. – URL: https://patents.google.com/patent/US7908520B2/en?oq=Self-testing+and-repairing+fault-tolerance+infrastructure+for+computer+systems.+Patent+US%2c+no.+7908520 (date of the application: 15.12.2020).

110. PT-CFI: Transparent Backward-Edge Control Flow Violation Detection Using Intel Processor Trace / Yu. Gu, Q. Zhao, Yi. Zhang, Z. Lin // CODASPY '17: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy (Scottsdale Arizona USA, March 2017). – Scottsdale, 2017. – P. 173–184. – URL: https://dl.acm.org/doi/abs/10.1145/3029806.3029830 (date accessed: 01.05.2017).

111. PTfuzz: Guided Fuzzing With Processor Trace Feedback / Gen Zhang, Xu Zhou, Yingqi Luo [et al]. – DOI: 10.1109/ACCESS.2018.2851237 // IEEE Access. – 2018. – Vol. 6. – P. 37302–37313. – URL: https://ieeexplore.ieee.org/document/8399803 (date accessed: 01.03.2019).

112. Randell, B. System structure for software fault tolerance / B. Randell. – DOI:10.1145/800027.808467 // IEEE Transactions on software engineering. – 1975. – Vol. SE-1, № 2. – P. 220–232.

113. Sklyar, V. V. Quality of Instrumentation and Control Systems: a Process

Approach. Lectures / Kharchenko V. S. (edit.) (date accessed: 08.03.2019). – Kharkiv: National aerospace university named after N. Zhukovsky «KhAI», 2013. 133 p. (date accessed: 08.03.2019).

114. The Art, Science, and Engineering of Fuzzing: A Survey / V. J. M. Manes, H. Han, Ch. Han [et al.]. – URL: https://arxiv.org/pdf/1812.00140.pdf (date accessed: 20.01.2018).

115. The FMEDA approach to improve the safety assessment according to the IEC61508 / M.Catelani, L.Ciani, V.Luong // Microelectronics Reliability Vol. 50, Issues 9–11 (Monte Cassino, September–November 2010). – 2010. – P. 1230–1235.

116. The STAR (self-testing and repairing) computer: An investigation of the theory and practice of fault-tolerant computer design / A. Avizienis, G. C. Gilley, F. P. Mathur [et al.] // IEEE Transactions on Computers. – 1971. – Vol. C-20 (11). – P. 1312–1321.

117. Timoc, C. C. Test Vectors Development and Optimization for a Mikroprocessor / C. C. Timoc, L. M. Hess, F. R. Stott // Autotestcon'80: International Automatic Testing Conference (Washington, DC, 2–5 November 1980). – 1980. – P. 165–170.

118. Tsai, T. An approach towards benchmarking of fault-tolerant commercial systems / T. K. Tsai, R. K. Iyer, D. Jewitt // Fault Tolerant Computing: proceedings of Annual Symposium (Sendai, Japan, 25–27 June 1996). − Sendai, 1996. − P. 314–323.

119. Using Simics and Simulation in IEC61508 Safety-Critical Systems – an Interview with Andreas Buchwieser / J. Engblom // Wind River Blog Network – URL: https://blogs.windriver.com/rwoolley/2014/11/using-simics-and-simulation-in-iec61508-safety-critical-systems-an-interview-with-andreas-buchwieser/ (date accessed: 07.01.2018).

# ПРИЛОЖЕНИЕ А. ПАТЕНТЫ НА ИЗОБРЕТЕНИЯ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

(19) **RU** (11) **2 697 629**(13) **C1**

(51) МПК
*G06F 11/26* (2006.01)

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

## (12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ПАТЕНТУ

(52) СПК
*G06F 11/261 (2018.08)*

(21)(22) Заявка: **2018105476, 13.02.2018**

(24) Дата начала отсчета срока действия патента:
**13.02.2018**

Дата регистрации:
**15.08.2019**

Приоритет(ы):
(22) Дата подачи заявки: **13.02.2018**

(45) Опубликовано: **15.08.2019** Бюл. № 23

Адрес для переписки:
**644009, г. Омск, ул. Масленникова, 231, АО "ОНИИП"**

(72) Автор(ы):
**Панков Денис Анатольевич (RU)**

(73) Патентообладатель(и):
**Акционерное общество "Омский научно-исследовательский институт приборостроения" (АО "ОНИИП") (RU)**

(56) Список документов, цитированных в отчете о поиске: **CN 104657247 A, 27.05.2015. RU 75070 U1, 20.07.2008. RU 2392657 C2, 20.06.2010. SU 1564628 A1, 15.05.1990.**

(54) **Устройство для имитации неисправностей в программно-аппаратных системах**

(57) Реферат:

Изобретение относится к вычислительной технике. Технический результат - расширение функциональных возможностей имитации неисправностей в программно-аппаратных системах. Устройство для имитации неисправностей в программно-аппаратных системах содержит внешнее устройство для имитации неисправностей, при этом в устройство дополнительно введены набор тестируемых микроконтроллеров, блок рабочей станции баз данных, которая хранит сигнатуры реакций системы, блок рабочей станции для имитации неисправностей, которая содержит экспертный алгоритм, предназначенный для поиска наиболее уязвимых мест программы и корректировки зоны применения алгоритма машинного обучения, и алгоритм машинного обучения для генерации данных для микроконтроллерной системы, устройство отладки по интерфейсу JTAG, блок обратной связи микроконтроллерной системы и блока рабочей станции баз данных. 1 ил.

RU 2697629 C1

RU 2697629 C1

РОССИЙСКАЯ ФЕДЕРАЦИЯ

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

(19) **RU** (11) **2 696 977**(13) **C1**

(51) МПК
*H04B 1/40* (2015.01)

## (12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ПАТЕНТУ

(52) СПК
*H04B 1/40 (2019.05)*

(21)(22) Заявка: **2018141019, 21.11.2018**

(24) Дата начала отсчета срока действия патента:
**21.11.2018**

Дата регистрации:
**08.08.2019**

Приоритет(ы):
(22) Дата подачи заявки: **21.11.2018**

(45) Опубликовано: **08.08.2019** Бюл. № **22**

Адрес для переписки:
**644009, г. Омск, ул. Масленникова, 231, АО "ОНИИП"**

(72) Автор(ы):
**Фомин Владлен Владимирович (RU),**
**Мартынов Андрей Валерьевич (RU),**
**Лушпай Александр Витальевич (RU),**
**Черненко Александр Валерьевич (RU),**
**Басов Павел Андреевич (RU),**
**Панков Денис Анатольевич (RU)**

(73) Патентообладатель(и):
**Акционерное общество "Омский научно-исследовательский институт приборостроения" (АО "ОНИИП") (RU)**

(56) Список документов, цитированных в отчете о поиске: **RU 85774 U1, 10.08.2009. RU 142937 U1, 10.07.2014. US 5859878 A1, 12.01.1999. RU 16948 U1, 21.03.2017. RU 2097919 C1, 27.11.1997.**

(54) **Носимая автоматизированная радиостанция диапазона КВ-УКВ**

(57) Реферат:

Изобретение относится к области радиотехники и может использоваться как унифицированное средство связи для персональной двусторонней дальней радиосвязи или в составе комплексов оборудования, так и для построения автоматизированных сетей абонентской радиосвязи с адресным доступом, интегрируемых в действующую инфраструктуру глобальных цифровых сетей с множественным доступом. Технический результат - обеспечение расширенных функциональных возможностей при одновременном повышении надежности радиосвязи в расширенном диапазоне частот. Носимая автоматизированная радиостанция диапазона КВ-УКВ содержит модуль антенно-фидерного тракта, радиочастотный модуль, состоящий в том числе из приемного радиотракта и тракта усилителя мощности и включающий переключатель режима, модуль управления и контроля, модуль цифровой обработки сигнала, модуль питания и зарядное устройство. 1 ил.

RU 2696977 C1

RU 2696977 C1

# ПРИЛОЖЕНИЕ Б. СВИДЕТЕЛЬСТВА О РЕГИСТРАЦИИ ПРОГРАММ ДЛЯ ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

**RU2019661730**

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

## ГОСУДАРСТВЕННАЯ РЕГИСТРАЦИЯ ПРОГРАММЫ ДЛЯ ЭВМ

Номер регистрации (свидетельства):
**2019661730**
Дата регистрации: **05.09.2019**
Номер и дата поступления заявки:
**2019660514 27.08.2019**
Дата публикации и номер бюллетеня:
**05.09.2019 Бюл. № 9**
Контактные реквизиты:
**нет**

Автор(ы):
**Панков Денис Анатольевич (RU),**
**Кондратьев Александр Николаевич (RU)**
Правообладатель(и):
**Акционерное общество «Омский научно-исследовательский институт приборостроения» (RU)**

Название программы для ЭВМ:
**Программа автоматизированного тестирования для поиска неисправностей портативного радиоприемного устройства**

**Реферат:**
Программа реализует диагностику программно-аппаратных неисправностей во время тестирования и работы устройства. Программа осуществляет: тестирование приема спутниковых, КВ и GSM- сообщений с использованием векторного генератора и USRP-передатчика; подготовку стратегий автоматического тестирования по времени; диагностику неисправностей для анализа результатов тестирования в автоматическом режиме. Тип ЭВМ: IBM PC-совмест. ПК. ОС: Windows.

**Язык программирования:** C++

**Объем программы для ЭВМ:** 13,3 Мб

РОССИЙСКАЯ ФЕДЕРАЦИЯ

**RU2019664742**

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

# ГОСУДАРСТВЕННАЯ РЕГИСТРАЦИЯ ПРОГРАММЫ ДЛЯ ЭВМ

Номер регистрации (свидетельства):
**2019664742**
Дата регистрации: **13.11.2019**
Номер и дата поступления заявки:
**2019663695 01.11.2019**
Дата публикации и номер бюллетеня:
**13.11.2019 Бюл. № 11**
Контактные реквизиты:
**нет**

Автор(ы):
**Панков Денис Анатольевич (RU),**
**Козьма Павел Алексеевич (RU)**
Правообладатель(и):
**Акционерное общество «Омский научно-исследовательский институт приборостроения» (RU)**

Название программы для ЭВМ:
**Специальное программное обеспечение тестирования и имитации для приемного и передающего трактов радиостанции**

**Реферат**:
Программа реализует тестирование приемного и передающего трактов радиостанции. Программа осуществляет: проверку приемопередающего тракта радиостанции; управление контрольно-измерительными приборами; имитацию событий для приемного и передающего трактов с целью проверки функционирования радиостанции и наличия отказов и сбоев при работе; ведение журнала событий. Тип ЭВМ: IBM PC-совмест. ПК; ОС: Windows.

**Язык программирования**: C++

**Объем программы для ЭВМ**: 1,03 Мб

# ПРИЛОЖЕНИЕ В. АКТЫ ВНЕДРЕНИЯ

УТВЕРЖДАЮ
Генеральный директор
АО «ОНИИП»
_____ В.А. Березовский
«20» 02 2021 г.

**Акт внедрения**

материалов диссертационной работы Панкова Дениса Анатольевича, представленной на соискание ученой степени кандидата технических наук

Настоящим актом подтверждается, что результаты диссертационной работы Д. А. Панкова в части повышения устойчивости к неисправностям программно-аппаратных устройств с помощью применения имитации неисправностей использованы АО «ОНИИП» при проведении ОКР «Бумеранг».

Начальник отдела 5, к.т.н. К. А. Сидоренко

Заместитель генерального директора
по научной работе, к.ф.-м.н. С. В. Кривальцевич