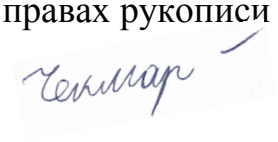
Министерство образования и науки РФ
ФГБОУ ВО «Сибирский государственный аэрокосмический университет имени академика М.Ф. Решетнёва»

На правах рукописи



Чекмарёв Сергей Анатольевич

МЕТОД ИНЪЕКТИРОВАНИЯ СБОЕВ ДЛЯ ТЕСТИРОВАНИЯ СБОЕУСТОЙЧИВЫХ МИКРОПРОЦЕССОРОВ ТИПА СИСТЕМА НА КРИСТАЛЛЕ

Специальность 05.13.05 – Элементы и устройства вычислительной техники и систем управления

Диссертация на соискание учёной степени кандидата технических наук

Научный руководитель:
кандидат технических наук, доцент
Ханов Владислав Ханифович

Красноярск – 2015

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность работы. Проектирование микропроцессоров, сбоеустойчивых к ионизирующему излучению космического пространства, является в настоящее время важнейшей задачей отечественных производителей микроэлектроники и электронного космического приборостроения. Различают несколько видов сбоев, происходящих в космосе с электронной аппаратурой. Наиболее частым сбоем микропроцессоров и систем на их основе являются одиночные сбои в памяти – во внутренней: в кэш-памяти и в файле регистров, а также во внешней памяти. На предотвращение сбоев в памяти направлены основные усилия по обеспечению сбоеустойчивости микропроцессоров.

Важным элементом создания сбоеустойчивого микропроцессора является отладка, тестирование и испытания тех или иных вариантов обеспечения его сбоеустойчивости. Тестирование и испытания должны удостоверить эффективность защиты памяти процессора от одиночных сбоев. Сбоеустойчивость процессоров в процессе тестирования подтверждается с помощью тех или иных методов внедрения (инъектирования) сбоев в его память – внутреннюю и внешнюю.

Методам инъекции сбоев и созданию на их основе систем тестирования элементов и устройств вычислительной техники и систем управления посвящены работы J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, A. Rajabzadeh, S.G. Miremadi, M. Mohandespour, A. Ejlali, A. Dasilva, J.-F. Martinez, L. Lopez, A.-B. Garcia, L. Redondo, B. Bastien, B.W. Johnson, D.D. Andres, J.C. Ruiz, D. Gil, P. Gil, T. Lenhart, Z. Haissam и другие. Методы тестирования устойчивости микропроцессоров к одиночным сбоям на основе инъектирования сбоев рассмотрены в работах M. Milko, C. Elks, R. Williams, J. Gaisler, M. Portela-Garcia, C. Lopez-Ongil, M. Garcia-Valderas, L. Entrena, A. Fidalgo, J. Ferreira, M. Gerigota, G. Alves. Известные методы обладают как достоинствами, так и недостатками. Некоторые являются универсальными,

другие зависят от возможностей тестируемой системы. Например, большинство современных микропроцессоров имеют внутрикристальный отладчик, позволяющий получить доступ к внутренним ресурсам. Эта возможность используются для модификации регистров, кэш-памяти и внешней памяти, обеспечивая полезный механизм для осуществления инъекций сбоев.

Существующие подходы к инъектированию сбоев, использующие возможности внутрикристальных отладчиков, обладают одним принципиальным ограничением: для проведения компаний по инъекции сбоев используются или внешний управляющий компьютер и/или внешний к тестируемой системе аппаратный отладчик, связанный с управляющим компьютером, так что подготовка и генерация инъекций для сбоев происходит вне тестируемой системы. Данная особенность усложняет систему инъекции сбоев, ограничивает количество вносимых сбоев и продолжительность компаний по их инъектированию, увеличивает время инъектирования, а также, в целом, снижает производительность компаний по инъекции сбоев. Кроме того, данные методы, так или иначе, воздействуют на всю целевую систему в целом, что несколько отдаляет эксперименты от реальных условий, когда сбои происходят непредсказуемо и независимо от внешнего аппаратно-программного окружения. К тому же, существующие подходы на основе внутрикристальных отладчиков не конкретизированы для микропроцессоров типа система на кристалле и не учитывают их возможности. Таким образом, актуальной является разработка метода инъектирования сбоев, предназначенного для микропроцессоров типа система на кристалле, без указанных выше недостатков.

Целью диссертационной работы является развитие технологии тестирования сбоеустойчивости с помощью инъектирования одиночных сбоев для микропроцессоров типа система на кристалле.

Для достижения цели были поставлены следующие **задачи**:

1. Исследовать существующие методы инъектирования одиночных сбоев в память микропроцессоров.

2. Разработать метод инъектирования одиночных сбоев в память микропроцессоров типа система на кристалле.

3. Разработать аппаратно-программную систему для инъекции сбоев в память микропроцессора типа система на кристалле.

4. Разработать методики проведения экспериментов по инъекции сбоев в память микропроцессора типа система на кристалле.

5. Реализовать предложенные решения в экспериментальном образце процессорного модуля для малого космического аппарата.

6. Провести экспериментальные исследования эффективности предложенных решений.

Объектом исследований являются технология тестирования микропроцессоров на сбоеустойчивость с помощью инъекций одиночных сбоев в память.

Предметом исследований являются метод, аппаратно-программная система и методики для инъектирования сбоев в память микропроцессора типа система на кристалле с помощью аппаратного блока внесения инъекций.

Методы исследований. Для решения поставленных в работе задач использовались методы проектирования цифровых элементов и устройств вычислительной техники и систем управления с помощью языков описания аппаратуры, методы тестирования микропроцессорной техники, методы проектирования систем на кристалле, методы теории помехоустойчивого кодирования, методы структурного программирования, методы планирования эксперимента.

Научная новизна.

1. Предложен новый метод инъектирования сбоев в микропроцессоры типа система на кристалле, отличающийся использованием встроенного аппаратного сложно-функционального блока инъекции сбоев, что позволяет

проводить автономные эксперименты по внесению аппаратных сбоев с минимальными временными задержками.

2. Предложена новая структура программно-аппаратной системы инъекции сбоев, использующая аппаратный блок инъекции ошибок, позволяющая проводить инъектирование сбоев, как во внутреннюю, так и во внешнюю память тестируемой системы.

3. Предложены новые методики проведения экспериментов по инъектированию сбоев с помощью аппаратного блока инъекции сбоев, позволяющие проводить с высокой скоростью автономные эксперименты с остановкой процессора для тестирования сбоеустойчивости внутренней памяти, а также с остановкой и без остановки процессора для тестирования сбоеустойчивости внешней памяти.

Практическая значимость заключается в разработке и создании аппаратно-программной системы инъектирования сбоев для тестирования на сбоеустойчивость микропроцессора LEON3, что позволило отработать и испытать функционал сбоеустойчивости процессорного модуля малого космического аппарата «ТаблетСат-Аврора». Результаты работы применимы для любых микропроцессоров типа система на кристалле, имеющих внутрикристальный отладчик, связанный с внутренней шиной. Разработанные решения могут быть использованы не только для тестирования сбоеустойчивости в лабораторных условиях, но и для диагностирования функционала сбоеустойчивости микропроцессора типа система на кристалле в условиях его штатной эксплуатации в составе бортовой системы управления космического аппарата.

Основные положения, выносимые на защиту.

1. Предложенный метод инъектирования сбоев с помощью аппаратного блока инъекции ошибок в микропроцессор типа система на кристалле позволяет проводить *автономные эксперименты* по внесению аппаратных

сбоев во внутреннюю и внешнюю память *с минимальными временными задержками*.

2. Предложенная структура программно-аппаратной системы инъекций сбоев *обладает низкой инвазивностью*, а разработанный сложно-функциональный блок для инъектирования сбоев *не требует больших ресурсов* для своей реализации.

3. Предложенные методики проведения экспериментов по инъекции сбоев с помощью аппаратного блока инъектирования сбоев *расширяют функциональные возможности* инъектирования сбоев и позволяют проводить с высокой скоростью автономные эксперименты с остановкой процессора для тестирования сбоеустойчивости внутренней памяти, а также с остановкой и без остановки процессора для тестирования сбоеустойчивости внешней памяти.

Достоверность изложенных в работе результатов обеспечивается непротиворечивостью с исследованиями других авторов; корректной реализацией инъектора сбоев для процессора LEON3 как в виде компьютерной модели, так и в виде аппаратного устройства на базе программируемой логической схемы; результатами экспериментальных исследований эффективности разработанной системы и их сравнением с существующими аналогами; проведенным внедрением результатов работы.

Внедрение результатов диссертационной работы. Результаты диссертационной работы использованы при выполнении ОКР по теме «Разработка аппаратуры информационного обмена бортового комплекса управления малого космического аппарата», договор от 01.10.2012 г. №31-12 с ООО «Спутникс»; используются при выполнении ПНИ по теме «Разработка бортового комплекса управления на базе технологии система на кристалле для цифровой платформы сверхмалого космического аппарата», Соглашение от 19.06.2014 г. № 14.574.21.0041 с Минобрнауки РФ, ФЦП «Исследования и разработки по приоритетным направлениям развития

научно-технологического комплекса на 2014-2020 годы», уникальный идентификатор RFMEFI57414X0041; а также в учебном процессе СибГАУ при обучении студентов по специальности 10.05.02 «Информационная безопасность телекоммуникационных систем» в курсе лабораторных работ по дисциплине «Схемотехника устройств цифровой обработки сигналов».

Апробация работы. Основные результаты работы докладывались на следующих конференциях: Региональная научно-техническая конференция «Системы обработки сигналов на базе ПЛИС и цифровых сигнальных процессорах», 2011г.; XVI Международная научная конференция «Решетнёвские чтения», 2012г.; Всероссийская научно-практическая конференция «Многоядерные процессоры, параллельное программирование, ПЛИС, системы обработки сигналов», 2013г.; III научно-техническая конференция «Разработка, производство, испытания и эксплуатация космических аппаратов и систем», 2014г.; XVIII Международная научная конференция «Решетнёвские чтения», 2014г.; XVIII Всероссийская научно-техническая конференция с международным участием «Современные проблемы радиоэлектроники», 2015г.; International Siberian Conference on Control and Communications (SibCon), 2015г.

Личный вклад. Основные научные результаты получены автором лично или совместно с Хановым В.Х. Работы по созданию системы инъекции сбоев и проведению экспериментальных исследований выполнены непосредственно автором.

Соответствие диссертации паспорту специальности. Диссертационное исследование соответствует области исследований специальности 05.13.05 по п.4 «Разработка научных подходов, методов, алгоритмов и программ, обеспечивающих надежность, контроль и диагностику функционирования элементов и устройств вычислительной техники и систем управления», согласно которому разработаны метод,

система и методика инъектирования сбоев в микропроцессор типа система на кристалле для тестирования его сбоеустойчивости.

Публикации. Основные результаты по теме диссертации изложены в 14 печатных работах, 4 из которых изданы в журналах, рекомендованных ВАК, 9 в материалах и тезисах докладов и 1 свидетельстве о регистрации программы для ЭВМ.

Объем и структура работы. Диссертация состоит из введения, четырех глав, заключения и трёх приложений. Полный объем диссертации 135 страниц текста с 27 рисунками и 31 таблицами. Список использованных источников содержит 102 позиции.

В **первой главе** рассматриваются основные причины сбоев аппаратуры, способы реализации механизмов сбоеустойчивости микропроцессоров, тестирование микропроцессоров на сбоеустойчивость с помощью инъекций сбоев; анализируются методы инъекции сбоев с помощью внутрикристального отладчика; рассмотрены особенности микропроцессоров типа система на кристалле; проводится постановка задачи исследования.

Вторая глава посвящена описанию разработанных метода, системы и методик инъектирования сбоев с помощью встроенного аппаратного инъектора сбоев во внутреннюю и внешнюю память микропроцессора типа система на кристалле.

В **третьей главе** представлена реализация предложенных решений по созданию системы для инъекций сбоев на примере программного процессора LEON3.

В **четвертой главе** приведены результаты экспериментальных исследований эффективности предложенных решений для тестирования сбоеустойчивости процессора LEON3.

ГЛАВА 1 ТЕСТИРОВАНИЕ СБОЕУСТОЙЧИВОСТИ МИКРОПРОЦЕССОРОВ

В первой главе рассматриваются основные причины сбоев аппаратуры, способы реализации механизмов сбоеустойчивости микропроцессоров, тестирование микропроцессоров на сбоеустойчивость с помощью инъекций сбоев; анализируются методы инъекции сбоев с помощью внутрикристального отладчика; рассмотрены особенности микропроцессоров типа система на кристалле; проводится постановка задачи исследования.

1.1 Сбоеустойчивость цифровых интегральных схем к ионизирующему излучению космического пространства

В настоящее время действие космического ионизирующего излучения рассматривается как одна из основных причин сбоев и отказов электронной аппаратуры [1], работающей как в земных условиях, так и, конечно, в условиях космоса. Радиационные эффекты, приводящие к сбоям и отказам, делятся на две основные группы [2, 3]:

- эффекты, связанные с изменением (деградацией) свойств полупроводникового материала интегральной схемы (ИС) в связи с накоплением разрядов под воздействием ионизирующего излучения (Total Dose Effects, TDF);
- локальные эффекты, вызванные воздействием на локальную область ИС высокоэнергетичных протонов (ВЭП) и тяжелых заряженных частиц (ТЗЧ), приводящие к немедленному сбою (Single Event Effects, SEE).

Деструктивное воздействие радиации разделяют на неустранимые (hard error) и устранимые отказы (soft error), т.е. сбои. К неустранимым отказам, связанным с физическим разрушениям полупроводниковой структуры, относятся разрушения транзистора (Single Event Burnout, SEB) и разрушение

изолятора (Single Event Gate Rupture, SEGR) полупроводниковой структуры ИС.

К сбоям относятся: кратковременные изменения уровня напряжения на противоположное на выходе логического элемента (Single Event Transient, SET); изменения состояния на противоположное запоминающего элемента (Single Event Upset, SEU) и одиночные сбои-защелкивания (Single Event Latchup, SEL) [4]. SET события кратковременные, проводящие к «ложным» импульсам в цепях логических элементов, аналогичные помехам по цепям питания и от источников внешних электромагнитных помех для устройств на печатных платах. SEU вызывают в массивах памяти одиночные (Single Bit Upset, SBU), а иногда и мультибитовые сбои (Multiple Bit Upset, MBU), которые можно устранить путем перезаписи в соответствующие ячейки памяти правильных значений. События SEL, имеющие еще название тиристорного эффекта, самые опасные из одиночных сбоев: при отсутствии средств защиты они могут стать причиной необратимого отказа.

В ряде исследований показано, что по сравнению с TDF подавляющее число отказов и сбоев электронной компонентной базы (ЭКБ) летательных аппаратов обусловлено одиночными событиями [5], среди которых события SEU наиболее распространены [6]. Рассмотрим методы защиты от SEE более подробно.

Сбоеустойчивость – это свойство системы, например микропроцессора, продолжать свое нормальное функционирование в случае одного или нескольких сбоев некоторых из своих компонентов. Сбоеусточивые системы маркируются символами FT (Fault Tolerant). Обеспечение сбоеустойчивости необходимое условие достижения приемлемого уровня надежности систем, работающих как в земных условиях, так и в условиях космического пространства.

Сбоеустойчивость систем, работающих в космосе, обеспечивается специальными проектными решениями, обеспечивающими защиту от

радиации (Radiation Hardening, RH). Такие системы называются RH-системы, соответственно, микропроцессоры – RH-микропроцессоры. Для ЭКБ, в том числе и для ИС, можно рассматривать RH с двух сторон [7]:

1) как RH-процесс создания базовых технологий формирования полупроводниковых структур, используемых для изготовления ЭКБ (Radiation Hardening by Process, RHBP). К RHBP-решениям относятся в первую очередь технология кремний на изоляторе (КнИ) и кремний на сапфире (КнС) [1]. Для ИС разработанных с применением данных технологий не характерны эффекты типа SEL [2].

2) как RH-методологию проектирования функционала ЭКБ (Radiation Hardening by Disign, RHBD). RHBD можно разделить на 3 подхода в зависимости от уровня сложности, с которого рассматривается ИС (рисунок 1.1):

- RHBD на архитектурном уровне (Radiation Hardening By Design at Architecture Level, RHBD-AL);
- RHBD на схемном уровне (Radiation Hardening By Design at Circuit Level, RHBD-CL);
- RHBD-на уровне компоновки (Radiation Hardening By Design at Layout Level, RHBD-LL).

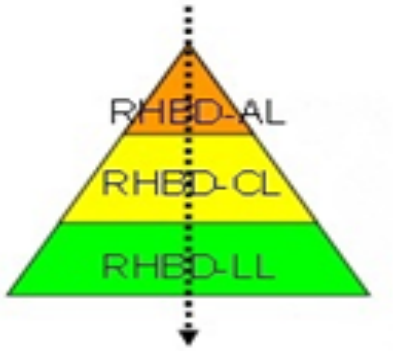


Рисунок 1.1 – Уровни RHBD [7]

На архитектурном уровне сбоеустойчивость обеспечивается: троированием функциональных блоков; особыми приемами проектирования

памяти, разделяемой на несколько независимых автономных массивов, применением помехоустойчивых кодов при проектировании сбоеустойчивой памяти [8] и другими проектными решениями.

К схемотехническим методам относятся методы построения транзисторных ячеек, имеющих повышенную устойчивость к SEE. Например, широкое распространение получили ячейки с дуальными потоками данных DDSL (Dual Data Stream Logic), предложенные в работе [9], а также триггерные ячейки типа DICE (Dual Interlocked Storage Cell) [10]. Существуют и другие решения, например, применение защитных вентилей [11] для защиты от SET.

RHBD-LL заключается в специальных приемах переноса и встраивания компонентов из библиотеки элементарных блоков (Rad Hard Libraries) в топологию каждого слоя ИС [7].

RHBD-методы повышения радиационной стойкости, как правило, приводят к увеличению энергопотребления и площади элементов, а также к ухудшению быстродействия [6]. Тем не менее, их применение является обязательным для ИС, эксплуатируемым в составе систем космического назначения.

Как чрезвычайно чувствительная к SEU, наиболее уязвимым элементом к космической радиации является оперативная память: внутренняя память микропроцессора (регистры и кэш), внешняя память, память реконфигурации FPGA типа SRAM. В современных системах оперативная память занимает большие объемы, что увеличивает вероятность SEU. Поэтому защита оперативной памяти является обязательной для аппаратуры космического назначения, в том числе и защита внутренней памяти микропроцессора. По существу архитектурная сбоеустойчивость микропроцессора заключается в защите его внутренней памяти от SEU, и является обязательным условием отнесения его к классу Fault Tolerant.

Основным способом защиты оперативной памяти является применение помехоустойчивых кодов. Функциональная схема использования помехоустойчивого кодирования памяти представлена на рисунке 1.2. В зависимости от типа помехоустойчивого кода он может автоматически обнаруживать и исправлять одну или две ошибки в памяти [12]. При этом они характеризуются разной степенью избыточностью проверочных бит, а также временными задержками на проведения проверочных и корректирующих действий. Кроме избыточности на выбор того или иного помехоустойчивого кода влияет срок работы электронной аппаратуры в условиях космоса: чем выше срок активного существования (САС), тем применяется более сложный код с большими возможностями по коррекции ошибок в памяти.

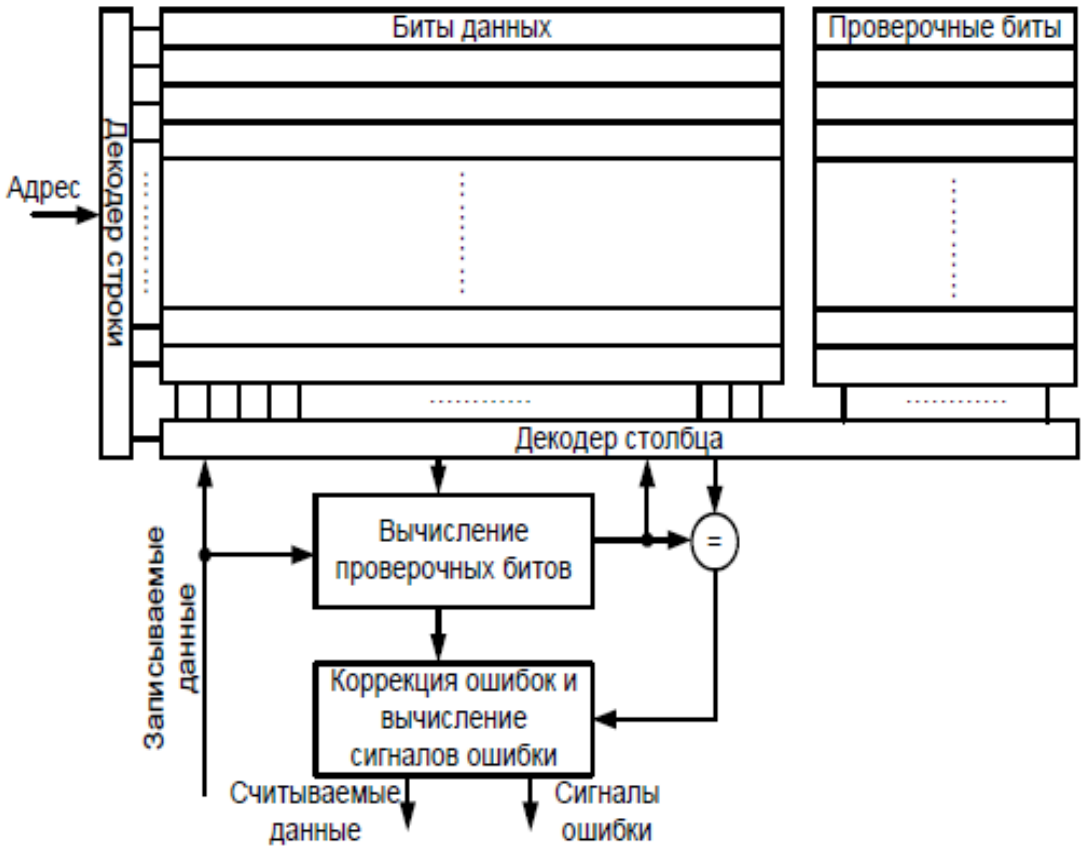


Рисунок 1.2 – Функциональная схема использования помехоустойчивого кодирования памяти [8]

Какими методами не обеспечивалась бы защита от одиночных сбоев важным является вопрос тестирования эффективности их применения для конкретных образцов ИС. Технологией оценки не чувствительности ИС к SEE и особенно к SEU является инъекция сбоев.

1.2 Тестирование сбоеустойчивости с помощью инъекций сбоев

Под технологией инъекций сбоев (Fault Injection, FI) понимается проверка надежности на основе реализации управляемых проверочных экспериментов, в которых наблюдается поведение тестируемой системы (System Under Test, SUT) в то время, когда в ней вызывают неисправности путем преднамеренного введения (инъектирования) сбоев [13]. Таким образом, в SUT вводятся сбои и наблюдается результат от их введения.

Технология FI уже в течение последних 30 лет используется с целью оценки отказо- и сбоеустойчивости во многий отраслях промышленности, связанных с эксплуатацией критически важных систем (КВС), в том числе и в космической индустрии. Международная электротехническая комиссия (МЭК, International Electrotechnical Commission, IEC) рекомендует использовать FI для определения последствий сбоев и их смягчения для КВС [14].

FI оказывает помощь в достижении двух взаимодополняющих основных целей [15]:

1) при тестировании системы;

2) для совершенствования проектных решений при ее создании.

Тестирование системы с помощью FI может включать две стадии:

- model-based testing, когда осуществляется моделирование инъекций сбоев в разработанную модель системы с целью определения адекватность модели вероятным сбоям и предсказание поведение в последующем реально созданной системы;

- coverage-based testing, когда в уже реализованную систему инъектируются сбои с целью реальной оценки механизмов детектирования сбоев и отказоустойчивости. Для этого применяются эксперименты, заключающиеся в осуществлении последовательности инъекций.

Технология FI расширяет понимание процессов введения сбоев в различные части целевой системы и улучшает понимание прогнозирования ее реакции на эти сбои, что оказывает влияние на выбор тех или иных проектных решений, которые касаются, например, процедур диагностики и самодиагности системы, т.е. ее контролепригодности.

В качестве формальной модели FI, определяющей применимость, понимание процесса инъекции сбоев и использования их результатов, принята модель FARM [13,16].

Для целевой системы, в которую вводятся сбои, определяются следующие наборы данных: входная область определяется набором сбоев *F* (Faults); функциональная область определяется набором выполняемых действий *A* (Activations); выходная область определяется набором результатов *R* (Readouts) и набором производных от них измерений M (Measures). Вместе наборы FARM-модели устанавливают главные атрибуты, которые позволяют полностью охарактеризовать FI. Эти атрибуты определяются следующим образом:

1) Множество сбоев *F*, которые преднамеренно вводятся в SUT. Каждый сбой характеризуется моделью сбоя *M* (одинарные или двойные, постоянные, периодические или кратковременные, устранимые или неустранимые и т.д. [17]); местоположением сбоя *L* (в регистрах, во внутренней или внешней памяти и т.д.); и временем сбоя *T* (случайным или периодическим, после наступления определенного события или исполнения определенной инструкции и т.д.). Множество всех сбоев называют пространством сбоев *F*, которое определяется как $M \times L \times T$ (рисунок 1.3). Основная проблема в определении конкретного подмножества F из всего

пространства неисправностей, которые могут быть инъектированы в разумные сроки и обеспечить статистически значимые результаты [18]. Часто F называют листом сбоев (Fault List).

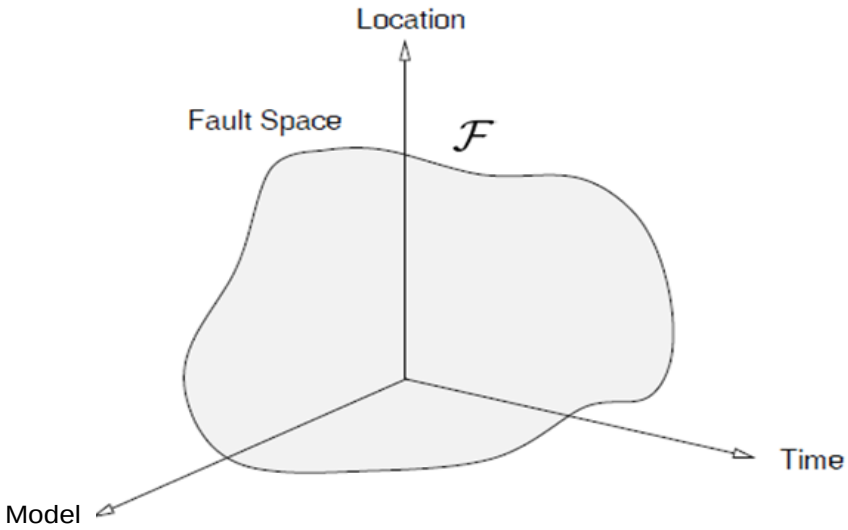


Рисунок 1.3 – Пространство сбоев [15]

2) Множество A определяет, какие функции выполняет SUT во время FI-эксперимента, например, принимает входные данные с датчиков, производит их обработку, выдает управляющие воздействия, информирует оператора о ходе работы и т.д. Для микропроцессорных систем в набор A добавляют набор W (Workloads) – программное обеспечение (ПО), выполняемое в ходе эксперимента и нагружающее процессор. Выбор A влияет на длительность каждого эксперимента, а также и на размер листа событий.

3) Множество R соответствует зарегистрированному поведению целевой системы во время FI-эксперимента. Данные сохраненные в R зависят от целевой системы и механизмов наблюдения за поведением SUT. Например, для микропроцессора сохраняемые данные могут включать результаты выполнения тестовой программы, данные памяти и регистров, данные аппаратных исключений, а также, при необходимости, временные диаграммы работы отдельных устройств процессора. Таким образом,

определение набора *R* является компромиссом между точностью и избыточностью эксперимента. Часто *R* называют Golden Result.

4) Множество *M* определяет величины, полученные в результате обработки экспериментальных данных, например, количество обнаруженных и из них исправленных сбоев, время задержки одного сбоя, оценка покрытия сбоями и т.д. для определения финальной оценки сбоеустойчивости целевой системы.

Для получения надежных оценок сбоеустойчивости проводится серия FI-экспериментов, часто называемые FI-компаниями. Каждый эксперимент имеет набор сбоев из множества *F*, выполняется при определенной функциональной нагрузке целевой системы из множеств *A* и *W*. В результате фиксируется набор результатов из множества *R*, которые обрабатываются для получения набора оценок из множества *M*.

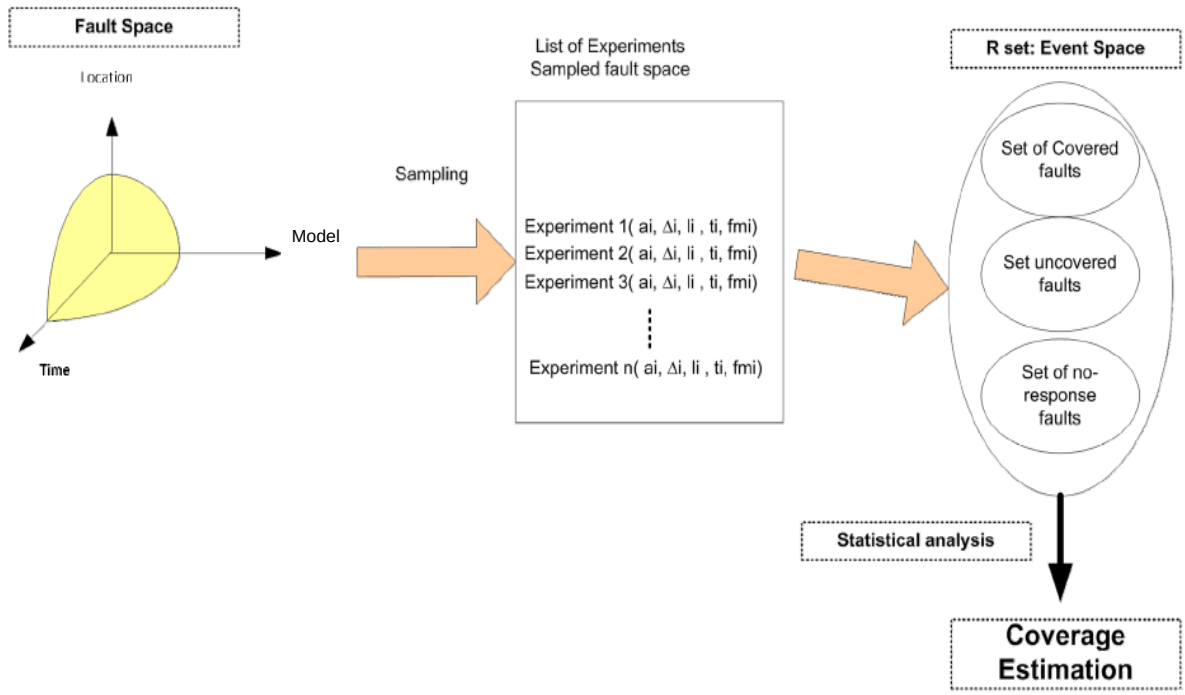


Рисунок 1.4 – Процесс проведения FI-экспериментов [15]

Данный процесс схематично представлен на рисунке 1.4, где *ai* – набор внешних воздействий, включая и нагрузочное ПО, *Δi* – длительность одной

инъекции, li – местоположение инъекции; ti – продолжительность эксперимента, fmi – тип сбоя, i – индекс эксперимента. Полученный набор результатов из множества R служит для получения оценок M с помощью статистического оценивания.

С практической точки зрения, критерий покрытия сбоев (Fault Coverage, FC) является важнейшей характеристикой любой FI-методологии. Он является мерой способности системы обнаруживать, изолировать и парировать сбои. Оценку критерия получают из результатов экспериментов, поэтому оценка принадлежит к множеству M модели FARM.

Если бы производилось исчерпывающее тестирование, то полученная в результате обработки оценка критерия покрытия сбоев приближалась бы к теоретически достижимому критерию. На практике идеальный случай получить невозможно из-за ограничений процесса тестирования. Всегда существует опасность, что инъектируемый сбой не был обнаружен целевой системой, например, при заданных условиях входных воздействий на целевую систему. А при других условиях, он был бы обнаружен. Чтобы получить хорошую точность оценки критерия покрытия сбоев необходимо увеличить количество экспериментов, проводимых в FI-компании.

Пусть n – количество инъекций произведенных в процессе FI-компании, в результате которого получено за время t общее количество сбоев $N(t)$. Полагая, что результат каждой инъекции Y соответствует распределению Бернулли, а вся совокупность результатов Y_1, Y_2, … Y_n биномиальному распределению получают, что статистическая оценка критерия покрытия сбоев равна [16]

$$\hat{C} = \frac{N(t)}{n}. \tag{1.1}$$

При $t \to \infty$

$$\hat{C} = \frac{1}{n}\sum_{i=1}^{n} Y_i, \tag{1.2}$$

где Y_i принимает значение 1 или 0.

Точность оценки критерия покрытия сбоев определяется по формуле

$$\widehat{S}[\hat{C}] = \frac{\hat{C}(1-\hat{C})}{n-1}, \quad (1.3)$$

а доверительный интервал

$$\left(\hat{C} - K_\gamma \sqrt{\widehat{S}[\hat{C}]}\right) < \dot{C} < (\hat{C} + K_\gamma \sqrt{\widehat{S}[\hat{C}]}), \quad (1.4)$$

где K_γ – квантиль нормального распределения, определяемый по значению доверительной вероятности (доверительного коэффициента, обычно 0,95).

Минимальное количество экспериментов, необходимое для получения достоверной оценки критерия покрытия сбоев вычисляется по формуле

$$n = \frac{\ln(1-\gamma)}{\ln C_l}, \quad (1.5)$$

где C_l – нижняя граница оценки критерия покрытия сбоев, γ – доверительный коэффициент.

В таблице 1.1 [15] представлена зависимость оценки критерия покрытия сбоев от числа экспериментов.

Таблица 1.1 – Зависимость оценки критерия покрытия сбоев от числа экспериментов [15]

n	C_l $\gamma = .80$	C_l $\gamma = .90$
100	0.983	0.977
1000	0.9983	0.9977
10,000	0.99983	0.99977
100,000	0.999983	0.999977
10^k	$0.9_{k-1}83$	$0.9_{k-1}77$

Из таблицы следует, что для получения оценки $\hat{C} = 0{,}999$ необходимо провести не менее 1000 экспериментов.

На рисунке 1.5 представлена блок-схема типичной архитектуры системы инъекции сбоев (СИС, FI-enviroment) [19]. Данная архитектура, конкретизированная для микропроцессора, соответствует FARM модели.

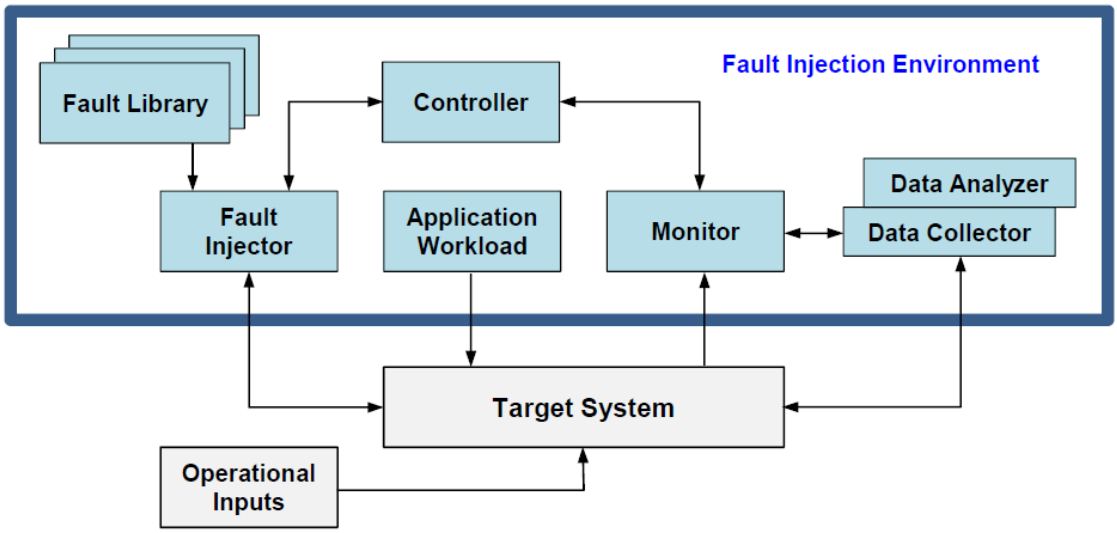


Рисунок 1.5 – Архитектура FI-enviroment [15]

Целевая система (Target System) испытывает воздействия FI. На вход (Operational Inputs) целевой системы поступает входная информация, которая обрабатывается соответствующим программным обеспечением (Application Workload). Библиотека сбоев (Fault Library) содержит лист сбоев, который задают требования для инъектирования сбоев в целевую систему с помощью инъектора сбоев (Fault Injector). Монитор (Monitor) глобально поддерживает функционирование целевой системы во время FI и при необходимости считывает из нее результаты инъекций. Накопитель (Data Collector) сохраняет полученные результаты и передает их для обработки в анализатор данных (Data Analyzer). Анализатор данных – автоматизированная система, с помощью которой даются оценки результативности FI и эффективности механизмов сбоеустойчивости целевой системы.

Несмотря на большое разнообразие конкретных реализаций FI данная архитектура справедлива для большинства СИС.

1.3 Методы инъекции сбоев

Технология FI включает множество подходов и направлений. В технической литературе можно найти несколько обзоров и классификаций FI, по существу очень близких друг к другу, например [20, 21, 15]. Блок-схема одной из известных классификаций представлена на рисунке 1.6 [15].

Методы разделяются на две большие группы: физические (Physical), которые реально воздействуют на SUT с помощью какого-то физического процесса с целью внесения сбоя; и основанные на имитации сбоев (Simulation based) путем внесения ошибок в различные уровни представления SUT: логическую модель, функциональную модель, поведенческую модель, модель инструкций. Кроме того имеются гибридные (Hybrid) в которых сочетаются Physical и Simulation based методы, среди который отдельно выделяют многочисленные методы инъекций с помощью моделирования на ПЛИС типа FPGA (FPGA Hardware Emulation), а также новейшие методы, классификация которых пока не проведена.

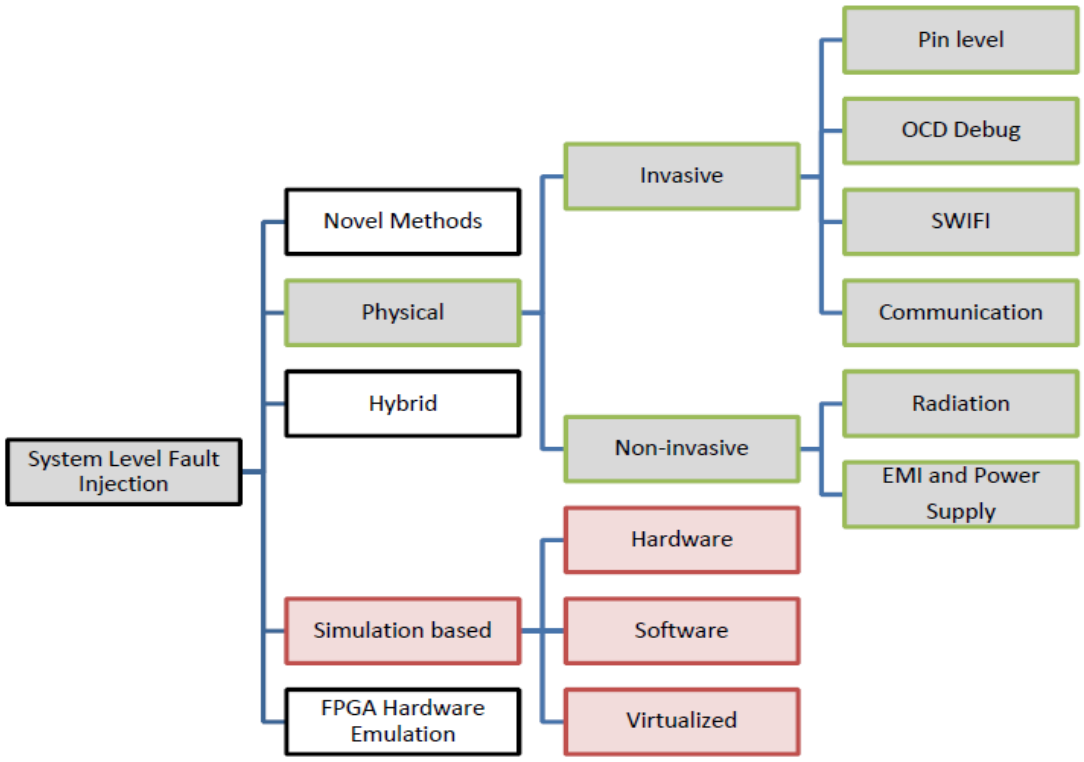


Рисунок 1.6 – Классификация методов FI [15]

Физические методы делятся на инвазийные и неинвазийвные, т.е. требующие какого-либо вмешательства в SUT или нет. Инвазийные методы обычно требуют аккомодации SUT к проведению FI-экспериментов. К неинвазийным методам относится методы, основанные на воздействиях радиации, электромагнитного излучения и помех по электропитанию.

Метод радиационных испытаний является традиционным для исследования сбоеустойчивости ЭКБ. Для их проведения требуются дорогостоящие установки: ускорители ионов и протонов, лазерные имитаторы с фокусированным излучением и изотопные установки на основе альфа- и спонтанного излучений [22, 23, 24]. Радиационные методы FI подходят в первую очередь для окончательных испытаний ЭКБ, а не для тестирования и отработки тех или иных архитектурных решений при проектировании RHBD-процессоров, потому что данные процессы требуют более высокую скорость проводимых работ.

Электромагнитное излучение (ЭМИ) является частой причиной сбоев цифровых систем. Для проверки чувствительности аппаратуры к ЭМИ проводят соответствующие эксперименты [25] Для их проведения используют специальные дорогостоящие реверберационные камеры [26]. ЭМИ-методы FI не отличаются высокой управляемостью и не позволяют контролировать эффективность локальных технических решений по сбоеустойчивости микропроцессора, например, только работу EDAC-механизмов внутренней памяти.

Метод FI с помощью помех и нарушений в работе источников электропитания тестируемой системы вследствие низкой воспроизводимости результатов и низкой управляемости процессом инъектирования сбоев используется редко. Обычно он используется как дополнение к другим методам инъекций сбоев для КВС [27, 28].

Инвазивные физические методы FI вносят в тестируемую систему те или иные изменения, которые требуются для их работы. Pin Level методы

требуют для инъекции специальные электрические пробники или сокеты через которые в SUT инъектируются сбои. Данные методы широко применялись в 80-90-ые годы [29, 30].

Методы, базируемые на возможностях внутрикристальной отладки (On Chip Debbuger, OCD) с помощью тестового порта (Test Access Port, TAP), например, Nexus, JTAG, BDM или OnCE, требуют их наличия в составе тестируемого микропроцессора. Впрочем, все современные процессоры в том или ином виде оснащены OCD-возможностями [31]. Подробнее OCD-методы рассмотрены в 1.5.

Широко распространен подход использования программного обеспечения (ПО) для инъекции сбоев (Software-implemented fault Injection, SWIFI). Методы SWIFI эмулируют аппаратные сбои с помощью ПО в аппаратные ресурсы доступные программисту. Они делятся на две группы: методы, в которых ПО генерирует и производит инъекции непосредственно во время работы тестируемой системы (run-time injection); и методы, в которых сбои подготавливаются заранее путем внесения изменений в исходный код программы или в ее бинарный образ перед ее загрузкой в память микропроцессора (pre run-time injection). Методы инъекций в реальном времени более распространены, так как они обеспечивают большие возможности для эмуляции аппаратных сбоев. Одновременно многие SWIFI-методы эмулируют и программные ошибки. Возможности конкретных SWIFI-методов и систем на их основе приведены в [32, 33, 34, 35, 36, 37, 38].

Как отмечено выше методы, основанные на имитации сбоев (Simulation based), инъектируют сбои в различные уровни представления SUT: логическую модель, модель функциональных блоков, поведенческую модель, модель инструкций или сразу одновременно на несколько модельных уровней. Для многих практических методов основой создания моделей тестируемых систем являются языки описания аппаратуры (Hardware Description Language, HDL), такие как VHDL и Verilog. Существуют

несколько подходов для реализации процесса FI, такие как модификация HDL-кода, модификация HDL-симулятора, управление симулятором с помощью скриптов [39,40].

Новые возможности для FI открыла технология полной системной симуляции или виртуализации. Эта технология комбинирует быструю модель инструкций тестируемой системы с ее точной логической моделью для создания виртуальной машины тестируемой системы. Наиболее известным продуктом полной системной симуляции является SIMICS [41], который может симулировать множество микропроцессорных систем таких как Alpha, x86-64, IA-64, ARM, MIPS (32- and 64-bit), MSP430, PowerPC (32- and 64-bit), POWER, SPARC-V8 and V9. Возможности SIMICS позволяют в режиме симуляции инъектировать сбои в тестируемую систему и исследовать ее чувствительность к ним [42].

Появление больших FPGA, позволяющих полностью вместить тестируемую систему, открыло новые возможности для тестирования их сбоеустойчивости. Сочетая преимущества программной разработки системы с помощью HDL с ее быстрой имплементацией в аппаратном устройстве, технология FPGA Hardware Emulation получила широкое распространение для FI. Одной из первых работ по данной тематике стала [43]. С тех пор появилось большое количество работ посвященных FPGA Hardware Emulation [44, 45, 46, 47, 48]. FI может быть осуществлена с помощью реконфигурации модели тестируемой системы путем внесения дополнительного модуля, осуществляющего FI, двумя способами: модуль вносится во время компиляции FPGA (compile time reconfiguration) или же модуль добавляется во время моделирования тестируемой системы (run-time reconfiguration). Второй случай применим посредством частичной реконфигурации FPGA, что возможно произвести для многочисленной группы FPGA типа SRAM-based.

Гибридные методы комбинируют несколько FI-технологий для достижения точности и объема проводимых экспериментов. Например, система NFTAPE позволяет использовать несколько типов инъекторов, используя методы SWIFI, FPGA Hardware Emulation и Physical Fault Injector [49].

В последнее время появилось ряд методов, которые невозможно отнести ни к одному из рассмотренных типов. К ним относятся формальные методы на основе построения различных математических моделей тестируемых систем [50, 51].

Среди рассмотренных методов лишь SWIFI и OCD-подход напрямую предназначены для тестирования микропроцессоров, используя их возможности. Но если SWIFI лишь программно имитируют аппаратные сбои, OCD-методы их реально производят. Рассмотрим методы основанные на OCD более подробно.

1.4 Инъекция сбоев с помощью внутрикристального отладчика микропроцессора

Средства отладки (On Chip Debbuger, OCD), включенные в состав большинства современных микропроцессоров, позволяют получить доступ к его внутренним ресурсам на чтение и для записи, причем без нарушения в работе работающих программных приложений. OCD инфраструктура обеспечивает доступ к внутренним ресурсам параллельно с целевым аппаратным обеспечением микропроцессора и работающим ПО. Эта возможность используется для FI.

В отличие от других FI-методов OCD-подход обладает рядом достоинств. Так, например, методы, основанные на имитации сбоев в модели, требуют больших затрат времени и могут привести к ошибочным результатам, поскольку зависят от качества модели. Методы SWIFI вносят

изменения в исполняемый код, что изменяет целевую систему; доступ к ресурсам ограничен только к тем, которые достигаются с помощью ПО. Большинство физических методов дороги и не позволяют точно управлять моментом и местом внесения сбоя.

Системы FI на базе OCD лишены указанных недостатков и поэтому довольно распространены. Примерами таких систем являются GOOFI (Generic Object Oriented Fault Injection) [52], INERTE (Integrated NExus-based Real-Time fault injection tool for Embedded systems) [53], Xception [54] и HiPeFI (High Performance Fault Injection) [55]. Большинство этих систем применяют внешние системы отладки (рисунок 1.7), обеспечивающие доступ управляющего компьютера к OCD.

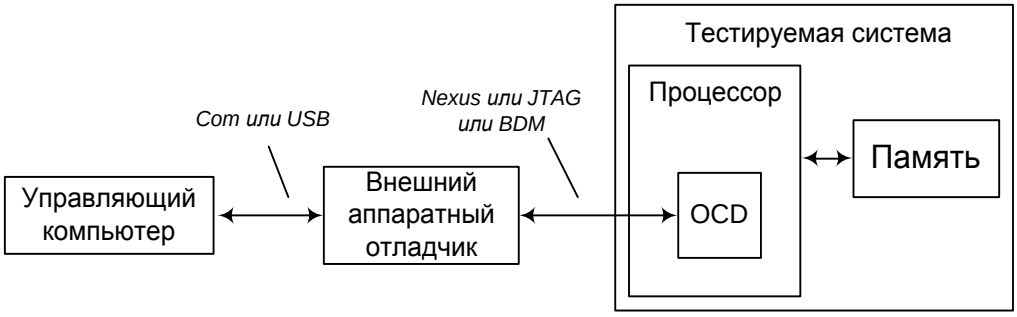


Рисунок 1.7 – Схема FI-системы на базе OCD

Инъекция сбоев через OCD, как правило, включает следующие шаги: установка точки прерывания и ожидание, когда текущая фоновая программа достигнет точки прерывания; при наступлении прерывания программы – чтение значения слова ячейки внутренней памяти, выбранной, например, случайным образом; изменение этого значения путем записи нового значения с внесенной, например, однократной битовой ошибкой; запись этого нового обратно в память; восстановление выполнения прерванной программы. Если в момент выполнения фоновой программы происходит обращение на чтение к ячейке внутренней памяти с внесенной ошибкой, то EDAC-механизм должен исправить данную ошибку. В случае обнаружения двойной ошибки

будет выдано исключение, которое может быть обработано ПО, и приняты соответствующие меры по обработке исключительной ситуации. Статистика о количестве внесенных и исправленных (или неисправленных) сбоев позволяет судить об эффективности защиты внутренней памяти микропроцессора. Доступ к OCD, как правило, осуществляется с помощью стандартизированных интерфейсов Nexus [56], JTAG [57] или BDM [58].

Рассмотрим несколько типичных примеров реализаций FI на базе OCD. В работе [59] представлена СИС, состоящая из управляющего компьютера, коммерческого отладчика и целевой системы на базе 32-разрядного микроконтроллера Freescale MC68332. Для взаимодействия коммерческого отладчика и целевой системы используется интерфейс BDM. Инъекции сбоев непосредственно выполняет внешнее по отношению к тестируемой микропроцессорной системе устройство отладки, однако, всю работу по подготовке каждого эксперимента и сохранению его результатов выполняет управляющий компьютер, что снижает производительность FI-компаний. Данный подход, когда используются обычные коммерческие отладчики, что определяет большую нагрузку по осуществлению FI-компаний на управляющий компьютер, встречаются достаточно часто [60].

В работе [61] представлена архитектура СИС, использующая стандарт IEEE JTAG 1149.1. Она также состоит из управляющего компьютера и тестируемой микропроцессорной системы на базе процессора ARM7 TDMI-S, имеющего OCD, а также специализированного устройства инъекции сбоев, вместо обыкновенного коммерческого отладчика. Устройство инъекции состоит из контроллера инъекции сбоев, накопителя планируемых сбоев, накопителя результатов инъекций, модуля команд отладчика и контроллера интерфейса JTAG. Данное решение позволяет функции управления инъекциями, подготовки планируемых сбойных событий и хранение промежуточных результатов инъекций сбоев, осуществлять не во внешнем управляющем компьютере, а с помощью специализированного устройства

инъекции сбоев. Достоинством является возможность проводить длительные автономные от управляющего компьютера эксперименты по многократному внесению одиночных сбоев, отсутствии инвазивности целевой системы, а также относительно малая временная задержка на внесение одного сбоя равная 2 мс/сбой. Недостатком данной архитектуры, как и предыдущей, является сложность системы проведения экспериментов по инъекции одиночных сбоев, для которой требуется внешнее специализированное устройство инъекции сбоев.

В работе [62], обобщающей более ранние работы данных авторов [63, 64], представлены несколько вариантов реализации СИС, реализующих FI в реальном времени, основанные на возможностях стандарта Nexus. СИС реализованы с помощью FPGA. В FPGA входят простой VHDL-процессор целевой системы, имеющий совместимый с Freescale MPC565 внутрикристальный OCD, и специализированный отладчик, имеющий дополнительный функционал для FI. Процессор не имеет кэш-памяти. Он и отладчик связаны интерфейсом Nexus Class 2+, который позволяет обращаться к памяти целевой системы не останавливая процессор [56] и, соответственного, без остановки работающего ПО. Собственно эта возможность и определяется авторами работы как инъекция сбоев в реального времени, по аналогии с отладкой в реальном времени, определяемой в Nexus Class 2+.

Однако данный режим (чтение/запись в реальном времени) просто реализуется только для внешней памяти, для регистров процессора его реализовать можно лишь с помощью модификации OCD путем добавления в него FI-модуля (рисунок 1.8). Кроме того для осуществления FI в регистры процессора, требуется модификация доступа к ним, чтобы предотвратить коллизии одновременного обращения к регистрам процессора и FI-модуля. Таким образом, данное решение является высоко инвазивным, что снижает его практическую ценность. Аналогичное по подходу решение представлено

в работе [65], где специально разработанный внутрикристальный отладчик, дополненный функционалом FI, имплементируется в процессор Intel 8051. Проект также как и последний пример, размещен в FPGA.

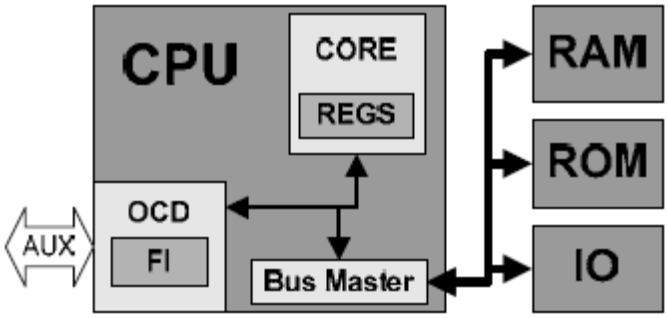


Рисунок 1.8 – OCD и модуль FI [66]

Инъекции сбоев в регистры производятся лишь в случае не обращения к ним со стороны процессора. Поэтому хоть производимая FI-компания не сказывается на текущую работу процессора, а сами инъекции имеют маленькую длительность, её общая производительность (количество инъекций в единицу времени) оказывается довольно низкой, что сказывается критерии покрытия сбоев. Поэтому в работе предложена *методика предопределенных инъекций*, когда инъекции производятся в заранее определенную ячейку памяти или регистр процессора при наступлении в работающей программе определенного события, например, достижения определенного места в программе или в случае появления в какой-то переменной определенного значения. В этом случае эффективность сбоя сильно возрастает. Противоположной является *методика случайных инъекций*, основанная на равномерном случайном инъектировании сбоев: инъекции случайны по времени, местоположению и по инъктируемому значению (однократный или двойной сбой). Данная методика приближена к естественному проявлению событий SEU и ее обычно используют для FI-компаний.

Вся работа [66] посвящена различным аспектам достижения реального режима инъектирования: реализации FI-процесса без остановки процессора и с остановкой процессора, сокращению задержки проведения одной инъекции, увеличению производительности FI-компании и сохранению при этом приемлемой инвазивности. Встает вопрос: насколько необходим реальный режим FI?

При FI без остановки процессора минимальное время инъектирования необходимо, чтобы освободить ресурсы, подвергаемые инъекциям, для процессора. Для FI с остановкой процессора задержки инъектирования влияют только на производительность FI-компаний. Поэтому реальный режим инъектирования необходим для FI без остановки процессора, но минимальная задержка инъектирования нужна в любом случае. При существующей базовой архитектуре процессоров реальный режим без инвазивности реализуем только для внешней памяти. Для реализации реального режима FI для внутренних ресурсов процессора нужно серьезное в него вмешательство.

Рассмотренные примеры не конкретизированы для процессоров типа система на кристалле (СнК), т.е. они, в общем, применимы к ним как к любому микропроцессору, обладающему OCD, но не учитывают их архитектурные возможности. Данные микропроцессоры обладают набором полезных особенностей, которые позволяют создавать эффективные СИС. Рассмотрим эти особенности СнК-процессоров подробнее.

1.5 Микропроцессоры типа система на кристалле

Технология СнК в полной мере отвечает основным требованиям электронного космического приборостроении – минимизации массы, габаритов и энергопотребления бортовой аппаратуры [67]. Вследствие этого СнК-микропроцессоры, как и собственно системы на кристалле (СнК, System on Chip, SoC) получили широкое распространение в электронном

космическом приборостроении. Они используются в системах полезной нагрузки [68, 69], но в первую очередь для создания бортовых компьютеров (БК, On Board Computers, OBC) [70, 71, 72].

СнК состоит из так называемых сложно-функциональных блоков (СФ-блоков), или как принято в англоязычной литературе IP-блоков (Intellectual Property). СнК-процессор содержит IP-блоки ядра процессора, внутрикристальной шины и большого набора контроллеров периферии. Модуль OCD обычно также является отдельным IP-блоком. IP-блоки являются унифицированными; унификация определяется типом внутрикристальной шины. Рассмотрим архитектуру типичного СнК-микропроцессора на примере микропроцессора UT699, основанного на софт-процессоре LEON3 [73] (рисунок 1.9).

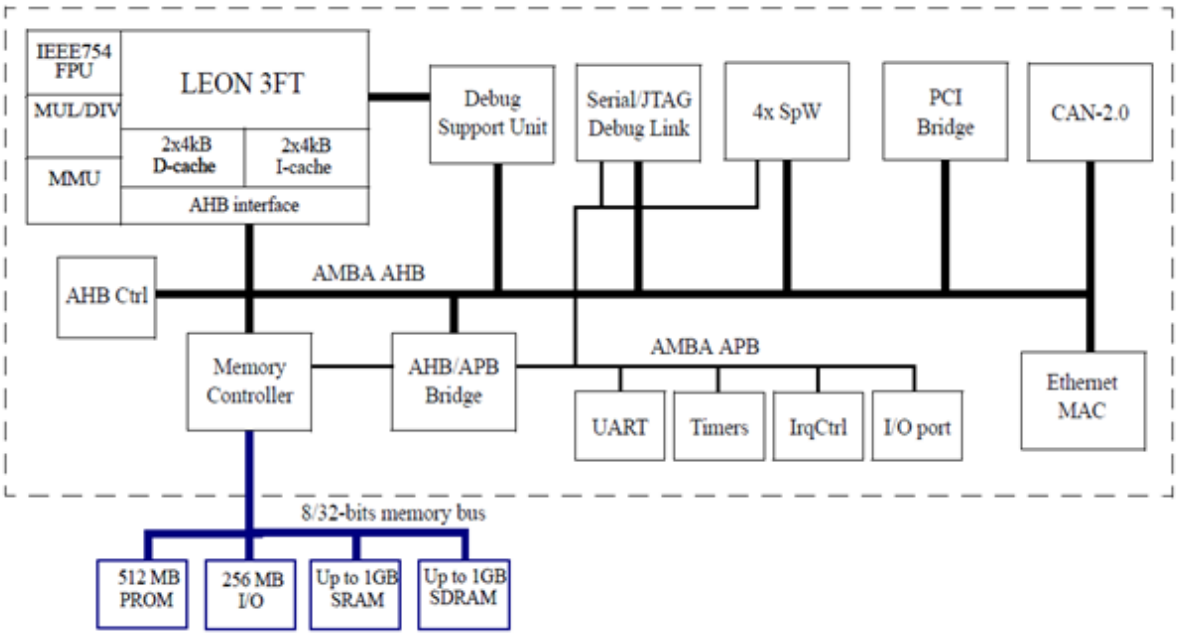


Рисунок 1.9 – Архитектура UT699 [73]

Микропроцессор UT699 LEON3FT базируется на IP-блоках из библиотеки Aeroflex Gaisler GRLIB [74]. IP-ядра подключаются к шине AMBA 2.0 [75] и конфигурируются в соответствии с правилами plug-and-play. Часть IP-блоков подключена к быстрой шине AMBA AHB, остальная

часть – к медленной шине AMBA APB. Модуль OCD носит название DSU (Debug Support Unit). Он также подключен к шине AMBA. Управляющий компьютер может подключиться к нему через интерфейсы RS232, PCI, JTAG, а таже SpaceWire, используя протокол RMAP [76].

Микропроцессор UT699 основан на отказоустойчивой версии LEON3FT. Как и в более ранней версии LEON2FT [77] в ней осуществляется защита внутренней памяти от ошибок SEU: регистры защищены помехоустойчивым кодом БЧХ(32, 7) [12], а кэш-память битами четности. LEON3FT является закрытой версией, но имеется открытая версия LEON3, доступная на уровне VHDL-кодов. Открытая версия не имеет встроенных механизмов сбоеустойчивости.

Для оценки чувствительности LEON3FT процессора [12], имплементированного в FPGA RTAX 2000, его разработчики использовали следующие способы FI: излучение готового микропроцессора в составе SUT тяжелыми заряженными частицами изотопа Калифорний-256; и инъекции ошибок SEU во внутреннюю память, используя внутрикристальный отладчик DSU.

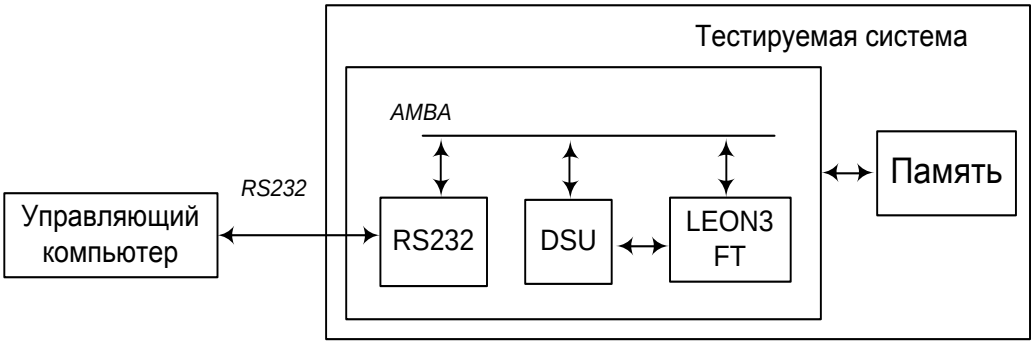


Рисунок 1.10 – Упрощенная FI-схема процессора LEON 3FT [78]

Упрощенная схема СИС представлена на рисунке 1.10. Инъекции планируются и осуществляются программно с помощью управляющего компьютера, который связан с DSU через RS232. Программа определяет вид и место инъекции, инициирует начало очередной инъекции, а после ее

проведения считывает ее результаты. Инъекции осуществляются с остановкой процессора. В отличие от схемы на рисунке 1.7 данная схема учитывает блочно-модульное построение СнК. Управляющий компьютер имеет связь с DSU без использования внешнего аппаратного отладчика. Подход не является инвазивным для целевой системы. Вместе с тем время осуществления каждого нового инъектирования зависит от работы внешнего управляющего компьютера, а, следовательно, будет в широких пределах изменяться от инъекции к инъекции, т.е. задержка инъектирования будет большой и переменной.

С другой стороны, как уже отмечалось выше в разделе 1.5, если FI-функционал перенести в сам тестируемый микропроцессор, можно достигнуть минимальных задержек и высокой скорости проводимых инъекций. Блочно-модульная архитектура СнК создает предпосылки для осуществления этого: необходимо модуль FI разработать как IP-блок и включить в СнК. Инъекции во время проведения FI-компаний будут вырабатывать в самом СнК-процессоре с временным задержками кратными его системным циклам, а управляющий компьютер будет выполнять роль просто накопителя результатов после проведения FI-компании. В этом, собственно, и заключается основная идея работы, целью которой является развитие технологии тестирования сбоеустойчивости с помощью инъектирования одиночных сбоев для микропроцессоров типа система на кристалле путём размещения FI-инфраструктуры инъектирования непосредственно в СнК-процессоре в виде специализированного IP-блока.

1.6 Выводы по главе и постановка задачи

В результате проведенного обзора можно сделать следующие выводы:

1. Обеспечение сбоеустойчивости является насущной проблемой обеспечения надежности современной ЭКБ, эксплуатируемой в условиях космического пространства.

2. Методы инъекции одиночных сбоев являются широко применяемым способом проверки чувствительности цифровых интегральных схем к SEU.

3. Инъекция сбоев с помощью внутрикристального отладчика является эффективным способом тестирования микропроцессоров к одиночным сбоям. Однако, OCD-методы воздействуют на всю целевую систему в целом, что несколько отдаляет эксперимент от реальных условий, когда сбои происходят непредсказуемо и независимо от внешнего аппаратно-программного окружения целевой системы.

4. Реальный режим может применяться для инъектирования сбоев во внешнюю память микропроцессора, реализация его для инъектирования во внутреннюю память требует серьезной переработки ядра микропроцессора.

5. Существующие подходы к инъекции сбоев с помощью внутрикристального отладчика не учитывают особенности микропроцессоров типа система на кристалле.

В соответствии с изложенным в данной работе поставлены следующие **задачи**:

1. Исследовать существующие методы инъектирования одиночных сбоев в память микропроцессоров.

2. Разработать метод инъектирования одиночных сбоев в память микропроцессоров типа система на кристалле.

3. Разработать аппаратно-программную систему для инъекции сбоев в память микропроцессора типа система на кристалле.

4. Разработать методики проведения экспериментов по инъекции сбоев в память микропроцессора типа система на кристалле.

5. Реализовать предложенные решения в экспериментальном образце процессорного модуля для малого космического аппарата.

6. Провести экспериментальные исследования эффективности предложенных решений.

ГЛАВА 2 ИНЪЕКЦИИ СБОЕВ С ПОМОЩЬЮ АППАРАТНОГО БЛОКА ВНЕСЕНИЯ ИНЪЕКЦИЙ В МИКРОПРОЦЕССОР ТИПА СИСТЕМА НА КРИСТАЛЛЕ

Вторая глава посвящена описанию разработанных метода, системы и методик инъектирования сбоев с помощью внутрикристального отладчика во внутреннюю память микропроцессора типа система на кристалле. Конкретизированы модель сбоев и требования к разрабатываемому подходу.

2.1 Модель сбоев и требования для инъектирования в микропроцессор типа система на кристалле

Как было ранее определено в главе 1 основной идей работы (она же является главным требованием к разработке) является размещение FI-инфраструктуры непосредственно в СнК-процессоре в виде специализированного IP-блока. Конкретизируем другие требования к разрабатываемому подходу, которые в первую очередь определяются принимаемой моделью сбоев.

Разрабатываемое решение предназначено для имитации сбоев в памяти процессора (внутренней и внешней), вызванных событиями SEU. Имитация SEU производится инвертированием значения в ячейке памяти. Виды сбоев – однократные, т.е. собственно SEU, и мультибитовые MBU – двукратные. Данные сбои наиболее типичные для микропроцессоров, работающих в космических условиях. Двукратные сбои могут быть реализованы с помощью одномоментного инвертирования двух соседних ячеек памяти или же с помощью внесения однократных сбоев с высокой скоростью, когда повышается вероятность получения ситуации, что последовательно произойдут ошибки в двух соседних битах памяти.

Таблица 2.1 – Модель сбоев и требования к разрабатываемому подходу

№	Модель сбоев	Требования к разрабатываемому подходу
1	*Тип имитируемого сбоя* – SEU.	*Требования к осуществлению сбоя*: - сбой должен быть реализован инвертированием значения ячейки памяти (триггера).
2	*Виды сбоев* – однократные или двукратные сбои.	*Варианты реализации*: - одномоментное инвертирование одного бита для имитации однократных сбоев или одномоментное инвертирование двух соседних битах памяти для имитации двукратных сбоев; - внесение однократных сбоев с высокой скоростью, когда повышается вероятность, что произойдут ошибки в двух соседних битах памяти.
3	*Направление инъектирования*: - внутренняя память (кэш и регистры); - внешняя память.	*Режимы работы СИС*: - с остановкой процессора в случае инвертирования во внутреннюю память; - без остановки процессора в реальном времени или с остановкой процессора в случае инвертирования во внешнюю память.
4	*Соответствие естественному для космического излучения покрытию сбоями*: - соответствующее естественному излучению космического пространства; - зависимое от состояния тестового ПО.	*Покрытие сбоями*: - всей памяти в случайное место и в случайное время; - при наступлении определенного события в тестовом ПО инъектирование в определенную ячейку памяти.

Основным режимом для внесения сбоев во внутреннюю память является режим с остановкой процессора. В этом случае не требуется специальной дополнительной аппаратной избыточности кроме IP-блока инъектора сбоев. Внесение сбоев во внешнюю память осуществляется в режимах, как с остановкой процессора, так и без остановки процессора. В этом случае также не требуется специальной дополнительной аппаратной

избыточности кроме IP-блока инъектора сбоев; FI осуществляется на основе штатных аппаратных средств СнК-процессора.

Покрытие памяти сбоями должно соответствовать естественному проявлению последствий SEU для космического излучения, т.е. равномерному случайному как по локализации, так и по времени. Кроме того, должен быть предусмотрен режим осуществления FI в определенные места памяти при наступлении в тестовой программе определенного события, например, при достижении переменной программы определенного значения. Данный режим является полезным при совместной отладке ПО и процессора на сбоеустойчивость.

Приведенные рассуждения сведены в таблицу 2.1. Кроме того, FI должны осуществляться с высокой скоростью, с минимальными задержками осуществления одного сбоя, FI компании должны быть статистически значимыми, а система для инъекций сбоев должна быть мало инвазивной, т.е. не допускать значительного дополнительного аппаратного вмешательства в процессор и использовать только его штатные (родные) средства.

2.2 Метод инъекции сбоев в микропроцессор типа система на кристалле с помощью аппаратного блока инъектирования

СнК-процессоры состоят из нескольких IP-блоков, включая и IP-блок ядра процессора, взаимодействующих с помощью внутрикристальной шины. Программные СнК-процессоры, специфированные с помощью языков описания аппаратуры, например, языка VHDL [79, 80], являются реконфигрируемыми. Их архитектура позволяет под конкретный проект удалять ненужные и добавлять необходимые программные (soft) IP-блоки. Идея предлагаемого метода инъекции [81, 82] использует данное свойство программных СнК-процессоров.

В отличие от аналогов инфраструктура инъекций сбоев как отдельный специализированный IP-блок располагается внутри СнК-процессора (рисунок 2.1). Внутрикристальный инъектор сбоев автономно осуществляет инъекции и собирает статистику чувствительности процессора к ним. Внешнему управляющему компьютеру отводится второстепенная роль: он только может учавствовать в сборе статистики через какой-либо внешний интерфейс.

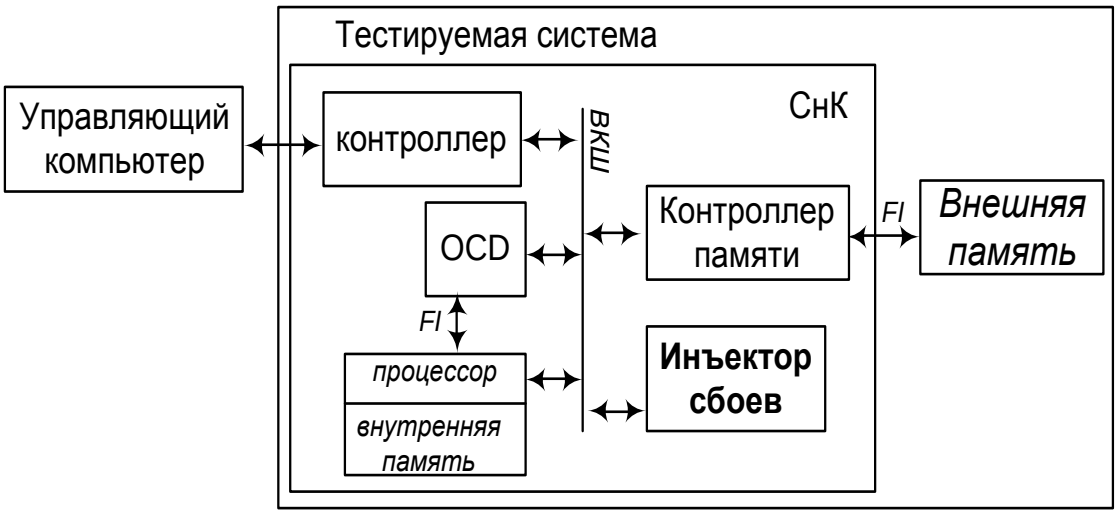


Рисунок 2.1 – Концептуальная архитектура FI-метода для СнК-процессоров

Инъектор сбоев может вносить сбои как во внутреннюю, так и во внешнюю память. При инъектировании во внутреннюю память инъекции производятся с остановкой процессора, но с минимальными предсказуемыми постоянными задержками, согласованными с временными циклами работы процессора. Поскольку задержки могут оказать существенное влияние на работу реальной системы при взаимодействии с различными внешними периферийными устройствами (например, работающими в синхронном режиме) невозможно проводить полноценную отладку микропроцессорной системы, если бы задержки были не постоянными и не предсказуемо изменяющимися в широких пределах.

Инъекции во внешнюю память производятся в реальном режиме работы процессора без его остановки в то время, когда процессор не обращается к внешней памяти [83].

Кроме того, для осуществления инъекций во внешнюю память доступен и режим с остановкой процессора. Он полностью соответствует режиму инъектирования для внутренней памяти с остановкой процессора.

Реализовать реальный режим для внутренней памяти без дополнительных аппаратных решений невозможно, что противоречит требованию низкой инвазивности. Данное обстоятельство рассматривается в разделе 2.3.2. В нашем же случае не требуется специальной дополнительной аппаратной избыточности кроме IP-блока инъектора сбоев; FI осуществляется на основе штатных аппаратных средств СнК-процессора.

OCD и инъектор сбоев расположены на внутрикристальной шине (ВКШ). Данное условие является обязательным для осуществления предлагаемой системы инъекции сбоев. Оно обеспечивает малую связность блока инъектора сбоев с ядром процессора. Взаимодействие обеспечивается только интерфейсом с ВКШ и не требует никакой дополнительной аппаратной избыточности. По окончанию процесса тестирования на сбоеустойчивость инъектор сбоев изымается из микропроцессора, не оставляя в нем никаких «следов». Таким образом, при данном подходе под низкой инвазивностью понимается не только малый размер модуля инъектора сбоев (что будет показано в главе 4), но и в первую очередь возможность добавления, а затем изымания IP-блока инъектора сбоев из микропроцессора по окончанию всех FI-экспериментов без какого-либо дополнительного вмешательства в аппаратные составляющие микропроцессора.

Рассматриваемая методология предполагает тестирование на сбоеустойчивость процессора, имплементированного вначале в FPGA. По окончанию тестирования и удалению инъектора сбоев, отлаженный

окончательный проект, может быть перенесен в более технологичные решения, например, в ASIC, а может быть оставлен в FPGA.

2.3 Аппаратно-программная система для инъекции сбоев в микропроцессор типа система на кристалле

Инъекции с остановкой и без остановки процессора имеют одинаковую систему для инъекций сбоев (FI-окружение, FI environment), но различаются ее режимами работы. Рассмотрим в отдельности работу СИС с остановкой процессора и работу СИС без остановки процессора. В обоих случаях подразумевается, что для предотвращения ошибок SEU в памяти (и внутренней и внешней) используется механизм EDAC на базе ECC.

2.3.1 Режим с остановкой процессора

Система для инъекций сбоев в режиме с остановкой процессора реализует алгоритм аналогичный осуществлению FI-инъекции с помощью OCD (см. раздел 1.4), с той лишь разницей, что процесс инъектирования определяет блок инъектора, а OCD используется как инструмент для доступа к памяти процессора. В разрабатываемом подходе СИС проводит FI как во внутреннюю, так и во внешнюю память. Архитектура СИС представлена на рисунке 2.2. Тестируемая система состоит из СнК-процессора, реализованного в FPGA, и его внешней памяти. С тестируемой системой связан внешний компьютер. СнК-процессор состоит из процессорного ядра, инъектора сбоев, контроллера внешней памяти, контроллера внешнего интерфейса. Процессорное IP-ядро соединено с OCD через отдельный интерфейс и оба блока подключены к ВКШ. Процессор и инъектор сбоев полностью независимы и связаны только ВКШ. Контроллеры внешней

памяти и внешнего интерфейса обеспечивают обмен с внешней оперативной памятью и управляющим компьютером соответственно.

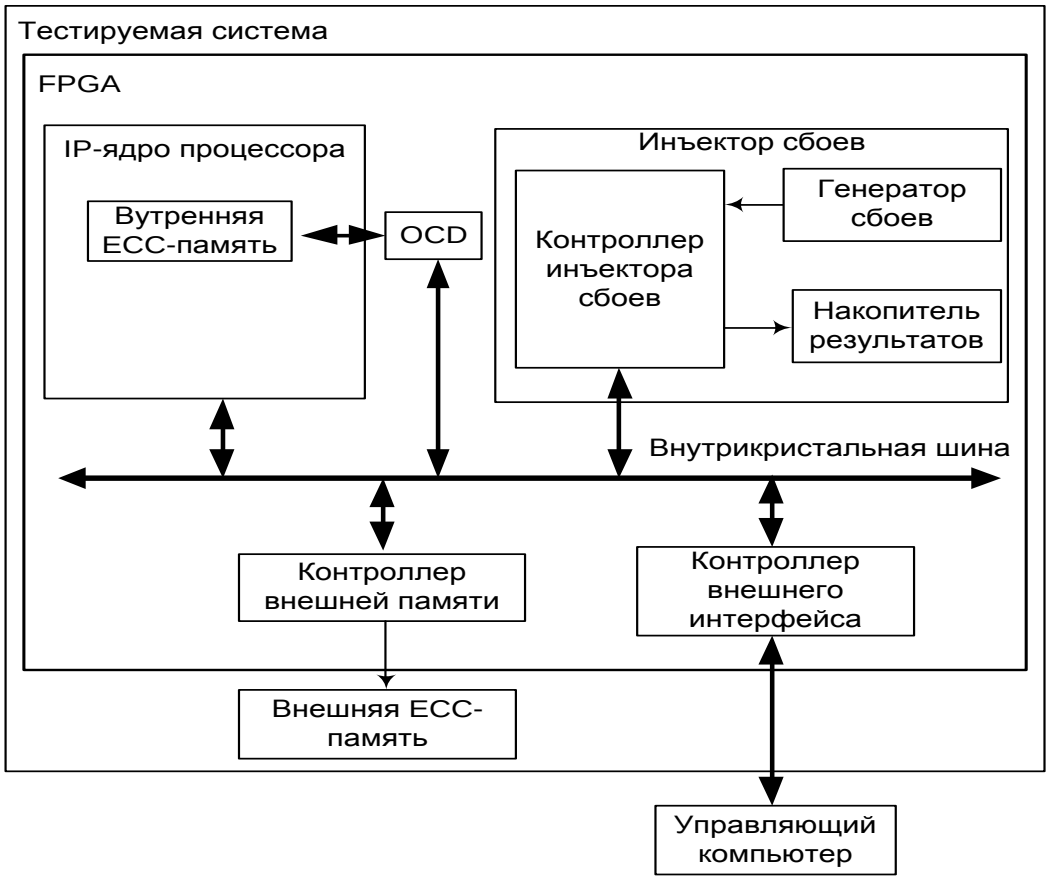


Рисунок 2.2 – Архитектура системы инъекции сбоев

Инъектор сбоев состоит из трех блоков: контроллера инъекции сбоев, генератора сбоев, накопителя результатов. Контроллер в целом координирует работу компонент инъектора сбоев; используя информацию, полученную от генератора, заносит ошибки в требуемые биты в требуемые адреса внутренней или внешней памяти; читает в регистре статуса EDAC-памяти информацию о факте обнаружения механизмом сбоеусточивости ошибки для проверки признака «ошибка исправлена/ не исправлена»; и сохраняет информацию о результатах внесения ошибки в накопителе.

Генератор вырабатывает (считывает) новый сбой для занесения во внутреннюю или внешнюю память из так называемого листа сбоев. Лист сбоев самостоятельно генерируется с помощью генератора псевдослучайных

чисел, являющегося частью генератора сбоев, или предварительно загружается из управляющего компьютера (рисунок 2.3). Для каждого одиночного эксперимента определяют четыре параметра:

– вид ошибки: однократная (ошибка в одном бите) или двукратная (ошибка в двух соседних битах);

– адрес ошибки в памяти;

– положение ошибки в 32-х битом слове памяти;

– момент времени для инъекции ошибки.

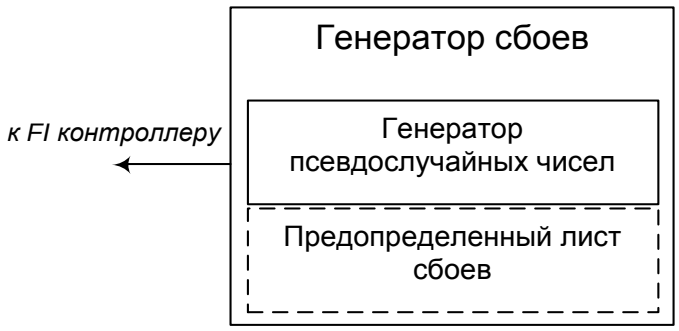


Рисунок 2.3 – Генератор сбоев

Накопитель результатов внесения ошибок сохраняет результаты эксперимента. Он имеет собственный блок памяти, который может быть достаточно большим, если FPGA имеет большой размер. В простейшем случае накопитель может содержать только количество обнаруженных ошибок и количество исправленных (или не исправленных) ошибок для каждого вида внутренней памяти процессора (кэш или регистровый файл) и внешней памяти. Для проведения более глубокого анализа результатов для каждого случая обнаружения ошибки в накопителе может сохраняться ее адрес в памяти процессора, если данную возможность имеет EDAC-механизм. В этом случае накопитель представляет таблицу (несколько таблиц), содержимое которой по окончанию FI-компании передается в управляющий компьютер. Если FI-компании рассчитана на большое количество экспериментов, то для предотвращения переполнения накопителя предусмотрена возможность его периодического считывания внешним

компьютером. Возможен также вариант использования под накопитель результатов экспериментов внешнюю память тестируемой системы, в то время как тестируется внутренняя память.

Как следует из описания, представленная структура предлагает множество вариантов использования штатных возможностей soft-процессора, имплементированного в FPGA. Ни каких дополнительных модулей кроме IP-блока инъектора сбоев не требуется. Не требуется также никого вмешательства в штатные модули процессора, например, OCD или ядро процессора, как это проводилось в [62]. Однако, реализация EDAC механизмов и внутренней и внешней памяти в части получения результатов сбоев должна быть согласована с предложенным подходом.

FI-компания проводится полностью автономно, без какого-либо внешнего управления. Только после ее окончания результаты передаются в управляющий компьютер. Это предопределяет высокую скорость проведения FI-компаний и малые задержки внесения сбоев. Возможности генератора сбоев позволяют проводить статистически значимые FI-компании, когда число экспериментов в одной FI-компании может достичь нескольких тысяч.

После завершения серии FI-компаний, подтвердивших сбоеустойчивость процессора, IP-блок инъектора сбоев изымается из проекта. Так как целостность остальных модулей в процессе тестирования сбоеустойчивости не была повреждена, процессор не нуждается в дополнительной повторной квалификации всех своих функций.

2.3.2 Режим без остановки процессора

В режиме без остановки процессора структура СИС не трансформируется (рисунок 2.2). Она по-прежнему включает дополнительный IP-блок инъектора сбоев в неизмененном виде. Однако теперь инъектор сбоев не останавливает процессор и соответственно

выполнение тестового ПО не останавливается. Такой режим в работе [62] назван инъектированием в реальном времени (real time FI): работа процессора над выполнением текущей программы и инъектирование сбоев происходит параллельно, независимо друг от друга.

При FI каждую инъекцию необходимо произвести за минимально возможное время, так как на это время внешняя память закрыта для обращения к ней процессором. Предлагаемое решение соответствует данному условию, т.к. временная задержка FI определяется только логикой работы ВКШ, обеспечивающей, по определению, очень быстрые транзакции по передаче данных.

Существует вероятность, что если процессор активно использует внешнюю память, то временная задержка между проведением соседних FI-экспериментов будет варьироваться в некотором небольшом диапазоне. Задержка определяется логикой работы контроллера ВКШ, который коммутирует обращение мастеров на шине к внешней памяти, в случае если процессор активно использует внешнюю память. Поэтому термин «real time FI», вероятно, не вполне корректен. В дальнейшем мы будем использовать определение «FI без остановки процессора».

Данный режим очень удобно использовать для отладки сбоеустойчивого ПО, особенно используя, так называемые, предопределенные инъекции, когда сбои проводятся в определенное место внешней памяти при наступлении в программе определенного события. Обсуждение этой возможности приведено в разделе 2.4.

Несмотря на привлекательность, реализация режима FI без остановки процессора для внутренней памяти процессора лишена практического смысла. Она потребовала бы существенного аппаратного вмешательства. На рисунке 2.4 представлен вариант такого решения для регистрового файла. Оно основано на применении менеджера коллизий.

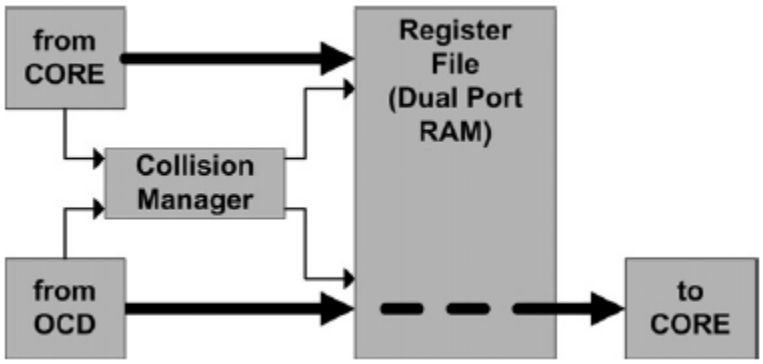


Рисунок 2.4 – Модификация регистрового файла [62]

Недостатков у данного решения можно указать несколько, но главным, перекрывающим все, является высокая инвазивность. Мы очень глубоко вмешиваемся в модули процессора, после чего его уже не возможно будет эксплуатировать без дополнительной квалификации его свойств и функций. Дополнительная квалификация должна ответь на вопрос: не нанесло ли такое вмешательство даже после удаления аппаратной избыточности к изменениям в работе процессора или к отклонениям от его базовых параметров?

Дополнительная квалификация очень трудоемкий и продолжительный процесс, поэтому она зачастую неприемлема на практике. Здесь еще раз отметим, что режим с остановкой процессора как для FI во внешнюю память, так и во внутреннюю память, дополнительную избыточность вносит только в виде отдельного IP-блока на ВКШ, который не требует переработки других модулей процессора и после удаления из проекта ничего в нем не оставляет. Поэтому процессор после окончания тестирования на сбоеустойчивость дополнительно квалифицировать не надо, он остался прежним, как и до тестирования. Режим без остановки процессора обладает такой же низкой инвазивностью только для FI во внешнюю память.

2.4 Методики для осуществления инъекций сбоев в микропроцессор типа система на кристалле

Испытания СнК-процессора на сбоеустойчивость состоят в проведении серии FI-компаний, состоящих из нескольких десятков, сотен и даже тысяч однотипных FI-экспериментов. Они позволяют проверить его чувствительность к сбоям при тех или иных входных условиях, например, с остановкой процессора или без остановки процессора, при разном количестве вносимых сбоев; при постоянной равномерной случайной или экспоненциальной интенсивности сбоев; при сбоях в связи с наступлением определенного события в тестовой программе или поступлении внешнего сигнала, при однократных сбоях или сбоях большей кратности и т.д. Сбои отдельно производятся в кэш-памяти, в файловом регистре и во внешнем ОЗУ.

Представленный метод может проводить разнообразные FI-компании высокой сложности. Каждая компания предварительно планируется, для нее разрабатываются свои тесты. В простейшем случае для каждой FI-компании создаются файлы описаний на языке Си. Си-файлы хранят тестовое ПО процессорного ядра и тестовое ПО управляющего компьютера.

В общем случае для каждой FI компании необходимо создавать свою конфигурацию инъектора (хранится в VHDL файле). В данной работе реализована возможность конфигурации инъектора сбоев с помощью ПО. В этом случае создается одна расширенная версия IP-блока инъектора сбоев, разработанная таким образом, чтобы оперативно реконфигурироваться под необходимый вариант FI-компании с помощью установки соответствующих битовых масок в управляющих регистрах инъектора. Данная реконфигурация может быть произведена тестовым ПО или с помощью дополнительного конфигурируемого ПО, например, по команде управляющего компьютера (рисунок 2.5). В первом случае тестовое ПО сначала конфигурирует

инъектор сбоев, дает ему указания для начала инъектирования, а затем выполняет нагрузочный алгоритм обращения к необходимому виду памяти. Во втором случае сначала конфигурационное ПО производит конфигурацию инъектора и завершается, а лишь за этим запускается тестовое ПО, которое при запуске дает инъектору команду для начала инъекций. Заметим, что несмотря на универсальность инъектора для всего разнообразия FI-инъекций, как это будет показано в главе 4, его размер остается довольно малым.

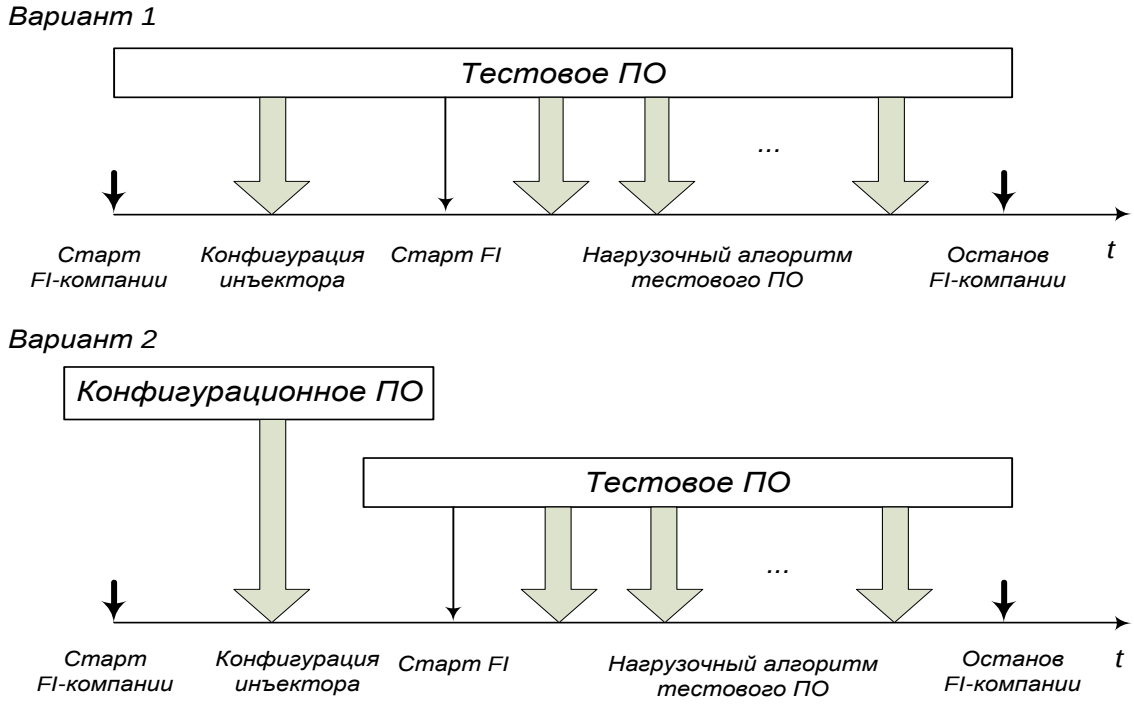


Рисунок 2.5 – Варианты осуществления переконфигурирования инъектора

Несмотря на разнообразие сценариев FI-компаний, существенные отличия определяются только FI-режимом: с остановкой процессора и без остановки процессора; а также способом покрытия сбоев: равномерным случайным (не предопределенным) или предопределенным.

Равномерное случайное покрытие сбоями приближается к естественному процессу возникновения SEU в элементах вычислительной техники под воздействием космической радиации. Данный способ покрытия

получил широкое распространение при тестировании микропроцессоров на сбоеустойчивость.

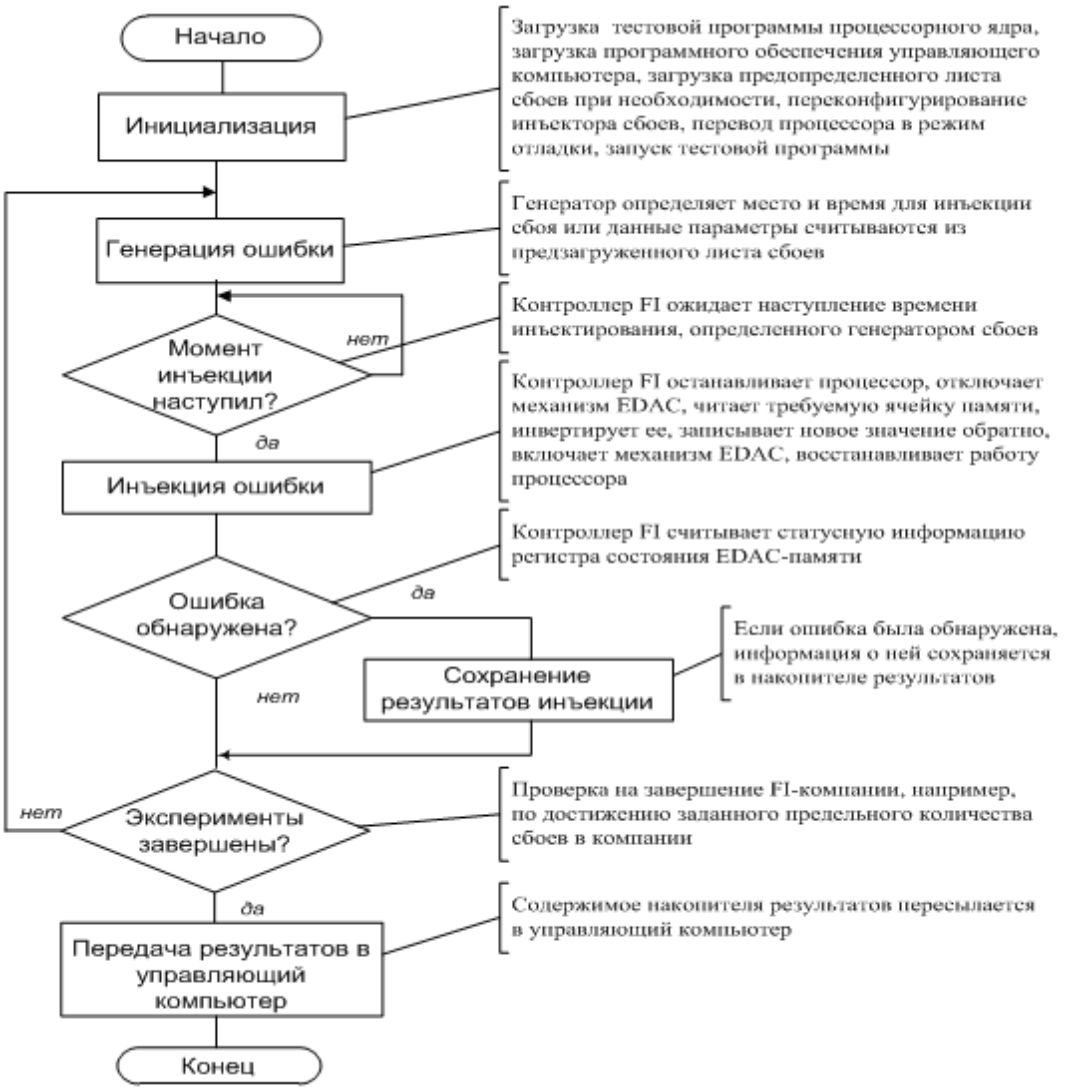


Рисунок 2.6 – Методика FI для случайного покрытия сбоев с остановкой процессора [81]

Предопределенное тестирование выполняется для совместной отладки сбоеустойчивости программного и аппаратного обеспечения. При предопределенном тестировании функции IP-блока инъектора усложняются. Он должен отслеживать состояние программы по данным, изменяемым в процессе ее работы, и, например, при обнаружении в определенной переменной ожидаемого значения производить инъектирование сбоя в заранее определенное место памяти. Таким образом, можно выполнить

проверку сбоеустойчивости программы в наиболее критические моменты ее работы. Значение места предстоящего инъектирования заранее известно, поэтому новое значение сразу записывается в память без предварительного чтения и инвертирования.

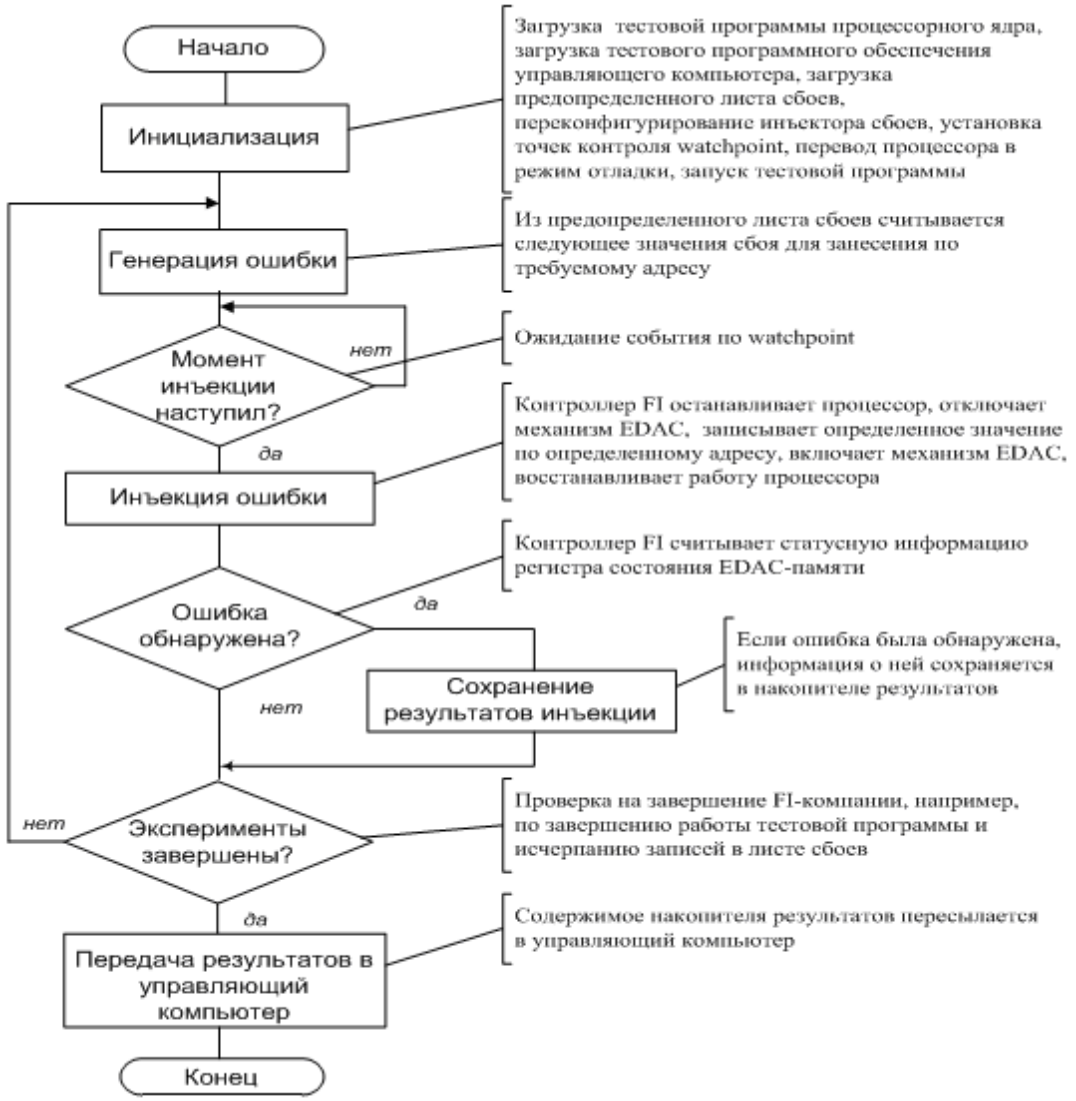


Рисунок 2.7 – Методика FI для предопределенного покрытия сбоев с остановкой процессора

Укрупненная блок-схема всех методик проведения FI-компаний остается одинаковой для всех способов FI, поддерживаемых разработанным подходом [81], а вот содержание отдельных этапов меняется существенно. Методики приведены на рисунках 2.6 - 2.9: на рисунке 2.6 приведена

методика FI для случайного покрытия сбоями с остановкой процессора, на рисунке 2.7 – методика FI для предопреленного покрытия сбоями с остановкой процессора, на рисунке 2.8 – методика FI для случайного покрытия сбоями без остановки процессора, на рисунке 2.9 – методика FI для предопределенного покрытия сбоями без остановки процессора.

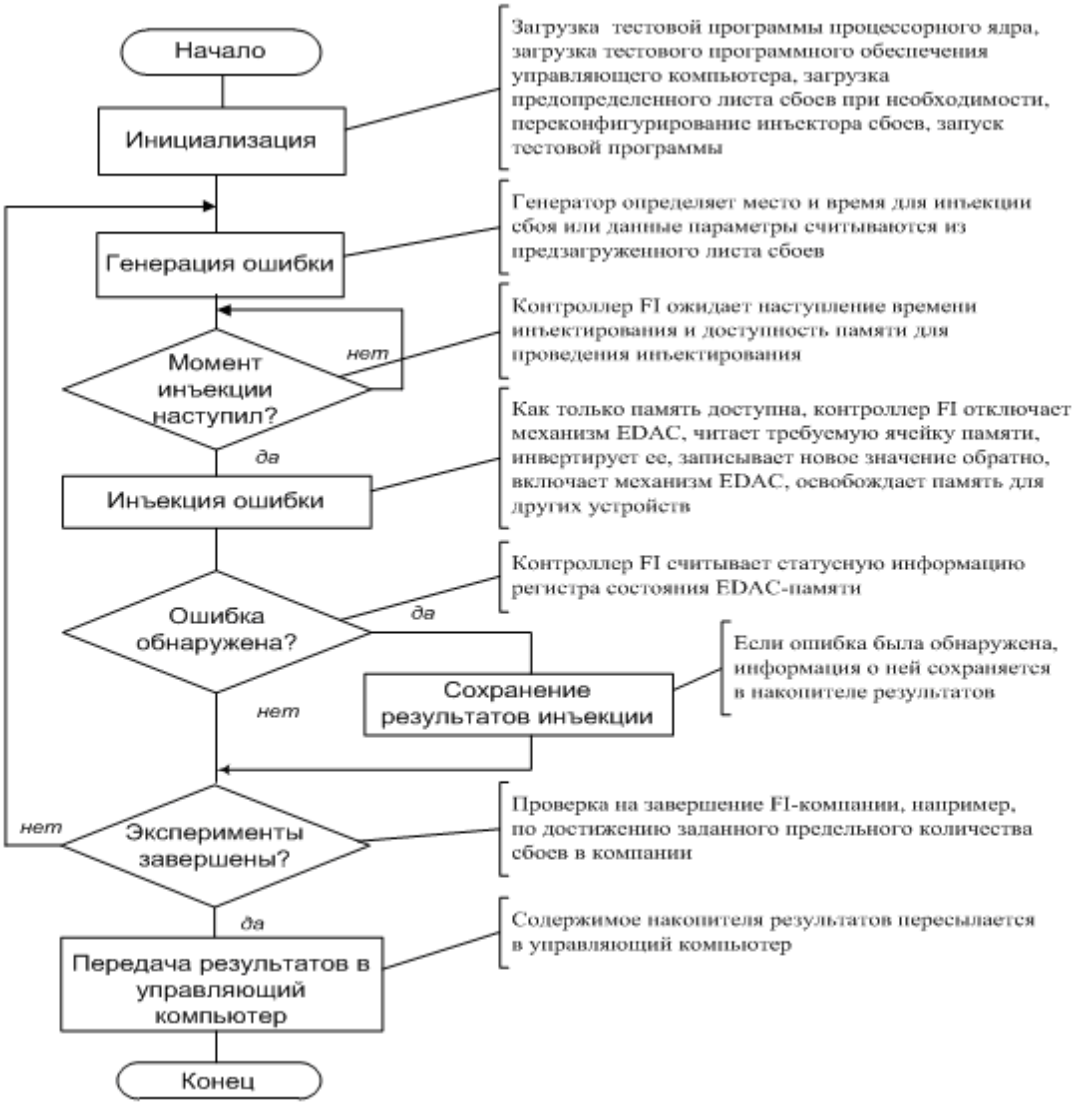


Рисунок 2.8 – Методика FI для случайного покрытия сбоев без остановки процессора

Процесс инициализации по большей степени включает одни и те же действия. В начале процесса инициализации это:

- загрузка тестовой программы, выполняемой тестируемым процессором. Ее основная цель выполнение нагрузочного алгоритма обращения к памяти для случайного равномерного покрытия или выполнения тестируемой на сбоеустойчивость конечной программной системы, например, бортового ПО;
- загрузка тестового ПО управляющего компьютера. Основная цель тестового ПО – считывание по окончанию FI-эксперимента всех его результатов, а затем их анализ и предоставление полученной информации инженеру тестирования;

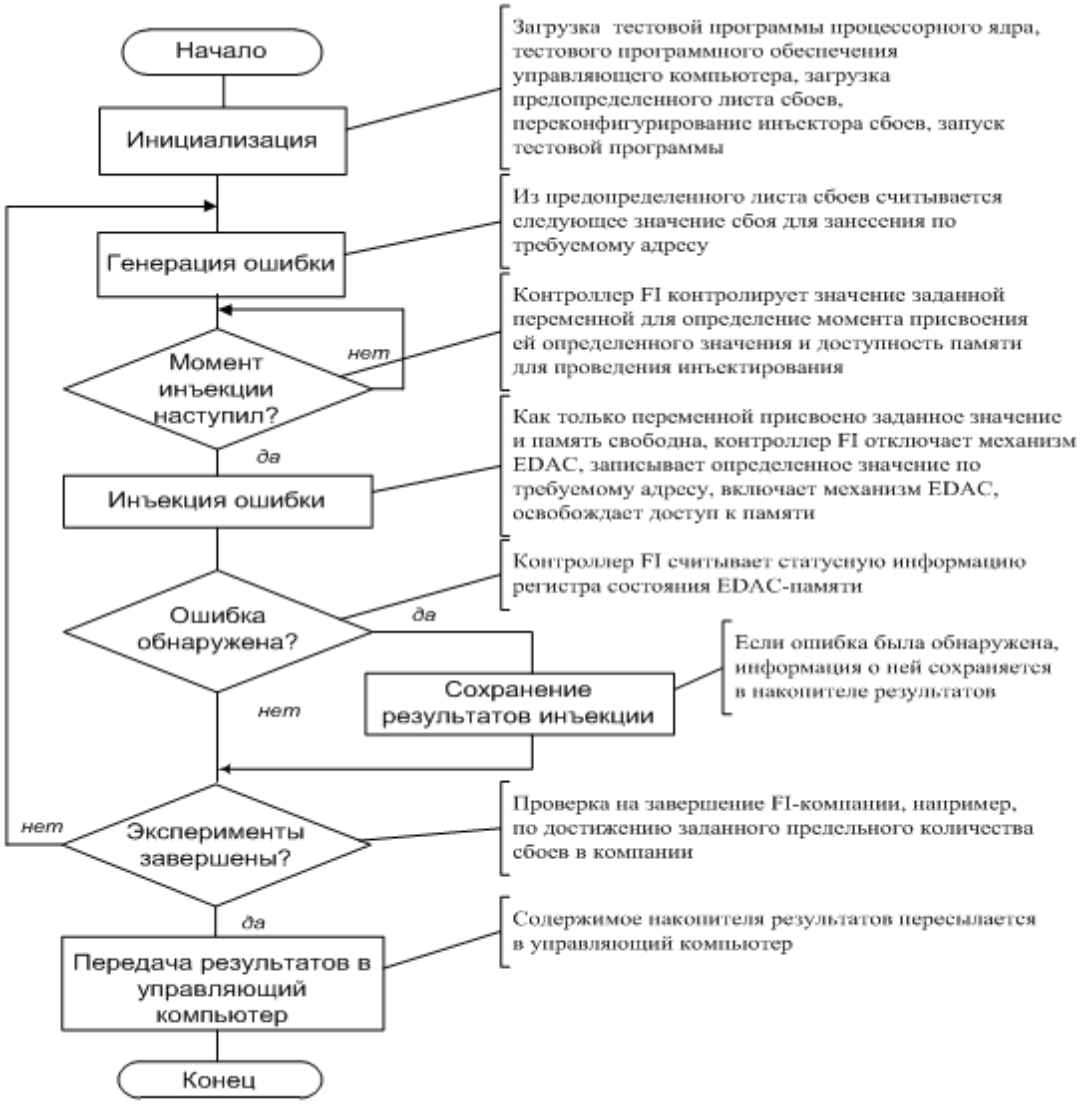


Рисунок 2.9 – Методика FI для предопределенного покрытия сбоями без остановки процессора

- загрузка предопределенного листа сбоев. Для предопределенного покрытия это действие обязательно, ведь именно из него контроллер FI узнает значение контролируемой переменной, в случае присвоения которого необходимо производить FI, а также значение переменной или ячейки памяти, в которую необходимо при инъектирование занести значение, также заранее определенное в листе сбоев. Для случайного покрытия лист сбоев загружается в том случае, если это требует сценарий FI-компании. Однако в данном случае параметры сбоев обычно определяются с помощью аппаратного генератора псевдослучайных чисел. Дело в том, для статистически значимого случайного покрытия количество инъектируемых событий должно быть достаточно большим. Для аппаратуры космического назначения, как следует из таблицы 1.1 необходимо провести не менее 1000 экспериментов. В этом случае размер листа сбоев составит 3,9 Кбайт (1000 слов по 32 бита каждое). Для некоторых типов FPGA, например, flash компании Microsemi, это величина достаточно большая. Поэтому проще генерировать сбои с помощью аппаратного генератора псевдослучайных чисел. Можно, конечно, уменьшить лист сбоев в несколько раз, а затем его данное количество раз последовательно использовать, чтобы получить общее количество сбоев 1000 или больше, но это усложняет реализацию;
- переконфигурирование инъектора сбоев под определенный сценарий проведения FI-компаний. Как это ранее обсуждалось, переконфигурирование может быть выполнено или с помощью специального конфигурационного ПО или с помощью тестового ПО, вначале его работы.

В конце процесса инициализации производится запуск нагрузочного тестового ПО.

Вот средние фазы инициализации отличаются. Для режима без остановки процессора – процессор не переводится в режим отладки и продолжает работу в обычном режиме. Для режима с остановкой процессора – процессор переводится в режим отладки, причем в случае

предопределенного покрытия – с предварительно установленными точками контроля watchpoint.

Генерация сбоев является процессом подготавливающим инъекцию. Он заключается в определении параметров предстоящего сбоя. Они читаются или из предварительно загруженного листа сбоев, или непосредственного генерируются генератором псевдослучайных чисел.

Время наступления момента для инъектирования определяется по-разному. В случае случайного покрытия сбоями оно определяется на этапе генерации сбоя, в качестве одного из его параметров. В случае предопределенного покрытия, оно определяется по watchpoint, или контроллер FI контролирует значение (snooping) заданной переменной для определения момента присвоения ей определенного значения.

Процесс непосредственного инъектирования для случайного покрытия сбоями происходит с предварительным чтением значения ячейки памяти, в чем нет необходимости при предопределенном покрытии сбоями, потому что данное значение уже известно перед началом инъектирования.

Все последующие этапы для всех методик одинаковы:

- контроллер FI считывает статусную информацию регистра состояния EDAC-памяти (внутренней или внешней), если ошибка была обнаружена, информация о ней сохраняется в накопителе результатов.
- если время окончания эксперимента не завершено, то происходит возврат к началу. В противном случае содержимое накопителя результатов пересылается в управляющий компьютер. Компьютер предоставит инженеру-тестирования полученные результаты для их сравнения с ожидаемыми.

При планировании FI-компаний необходимо учитывать, что не каждая внесенная в память ошибка будет обнаружена. Ячейка памяти с внесенной ошибкой может быть переписана процессором до того, как тестовое ПО процессора обратится к ней на чтение, и что вызовет запуск механизма

сбоеустойчивости. Задача нагрузочного ПО увеличить количество эффективных ошибок путём интенсивного обращения к памяти на чтение. Ошибки, которые привели к срабатыванию механизма сбоеустойчивости, называют эффективными. О чувствительности процессора к сбоям можно судить по коэффициенту чувствительности к сбоям *К*ч [84], который равен отношению неисправленных эффективных сбоев, вызвавших ошибки к общему *N*ош, к общему количеству эффективных сбоев *N*сб сбоев:

$$К\text{ч} = \frac{N\text{ош}}{Nс\text{б}}. \qquad (2.1)$$

Как следует из (1.1) и (1.2), коэффициент чувствительности к сбоям соответствует критерию оценки покрытия сбоями. Если все эффективные ошибки исправлены, то *Кч=0*, т.е. процессор не чувствителен к сбоям.

Для оценки вероятности не наступления катастрофического отказа *P(t)* под воздействием потоков частиц космической радиации в отечественной нормативной документации, используется следующее выражение [85]:

$$P(t) = e^{-\nu * t}, \qquad (2.2)$$

где ν – полная частота (количество событий в единицу времени) возникновения катастрофических отказов, определяемая в общем случае суммой частот возникновения отказов при воздействии тяжелых заряженных частиц, протонов и нейтронов с учетом мер принятых к их предотвращению , t – время, в течении которого микропроцессор находится во включенном состоянии. Нетрудно показать, что при $\nu \longrightarrow 0$ вероятность $P(t) \longrightarrow 1$, то есть чем меньше будет частота катастрофических отказов, тем меньше будет вероятность отказа микросхемы.

Представленный подход при равномерном случайном распределении инъекций сбоев позволяет оценить частоту катастрофических отказов микропроцессора при увеличении частоты одиночных сбоев памяти (ν_{seu}) на уровне его архитектуры. Иными словами, при эффективности принятых мер предотвращения сбоев при увеличении ν_{seu} возможно подтвердить, что $\nu = const \longrightarrow 0$.

2.5 Возможности контролепригодности

По окончании всей запланированной серии FI-компаний, подтвердившей удовлетворительные характеристики испытанного механизма сбоеустойчивости СнК-процессора, из него удаляют инъектор сбоев. В дальнейшем испытанный на чувствительность к сбоям процессор применяют либо в виде FPGA-реализации либо его имплементируют в ASIC.

Однако IP-блок инъектора сбоев может быть сохранен в окончательном варианте бортового компьютера с целью осуществления контролепригодности СнК-процессора. Для этого в генератор сбоев заносится предопределенный лист событий небольшого размера. В требуемый момент времени, по команде с наземного комплекса управления в СнК-процессоре процессор переводится в режим отладки и начинает работать инъектор сбоев. Предопределенные сбойные события вызывают ошибки в заданных местах внутренней памяти процессора. По окончанию процедуры диагностики результаты инъекций передаются на Землю. По их совпадению с ожидаемыми результатами судят о работоспособности механизма сбоеустойчивости СнК-процессора.

Данный режим хорошо подходит для FPGA-реализации, к тому же, как будет показано в главе 4, избыточность предложенных решений минимальна: она практически не влияет на ресурсы доступные для размещения процессора, так и на максимально возможную тактовую частоту процессора, имплементированного в FPGA.

Данную возможность для большинства космических миссий можно отнести к разряду второстепенных, учитывая низкую пропускную способность канала телеметрии отечественных КА. Гораздо больший практический интерес могла бы вызвать возможность подсчета реальных сбоев во внутренней памяти, вызванных ионизирующим излучением

космического пространства. До сих пор данная информация в открытых источниках фактически отсутствует. Между тем она имеет большую практическую значимость для разработчиков RH-микропроцессоров: насколько эффективны применённые меры защиты от ТЗЧ и активных протонов на практике?

Наработки представленного подхода открывают пути осуществления данной возможности. Для этого можно просто отключить режим инъектирования, и оставить только сбор статистики, которая сохраняется в накопители результатов. В остановке процессора при этом нет необходимости, процесс контроля сбоев будет осуществляться в реальном времени. Реализовать данную возможность можно на малых космических аппаратах, зачастую использующих в качестве элементной базы FPGA для размещения процессора. Однако рассмотрение данной темы не входит в круг задач данной работы.

2.6 Выводы по главе

Результаты проведенных исследований позволяют сделать следующие выводы:

1. Предложен метод инъектирования сбоев в микропроцессоры типа система на кристалле, отличающийся использованием аппаратного сложно-функционального блока инъекции сбоев, что позволяет проводить автономные эксперименты по внесению аппаратных сбоев с минимальными временными задержками.

2. Предложена структура программно-аппаратной системы инъекции сбоев, использующая аппаратный блок инъекции ошибок, позволяющая проводить инъектирование сбоев, как во внутреннюю, так и во внешнюю память тестируемой системы.

3. Предложены методики проведения экспериментов по инъекции сбоев с помощью аппаратного блока инъекции ошибок, позволяющие проводить с высокой скоростью автономные эксперименты с остановкой процессора для тестирования сбоеустойчивости внутренней памяти, а также с остановкой и без остановки процессора для тестирования сбоеустойчивости внешней памяти.

4. Разработанные методики поддерживают режимы инъекций с остановкой процессора и без остановки процессора со случайным равномерным и предопределенным покрытиями сбоев внутрикристальной и внешней памяти.

ГЛАВА 3 СИСТЕМА ДЛЯ ИНЪЕКЦИЙ СБОЕВ ДЛЯ МИКРОПРОЦЕССОРА LEON3

В **третьей главе** представлена реализация предложенных решений по созданию системы для инъекций сбоев на примере программного процессора LEON3.

3.1 Исходные данные для разработки

Процессор LEON3 является типичным СнК-процессором. Выбор процессора LEON3 для апробирования предложенного подхода объясняется простым обстоятельством: он был определен техническим заданием в качестве процессора бортового компьютера (процессорного модуля, ПМ) для малого космического аппарата «ТаблетСат-Аврора», в разработке которого принимал участие автор данной работы [86, 87]. Разработка ПМ проводилась в 2012-2013 гг. – экспериментальный образец; в 2013-2014 гг. – летный образец. ТаблетСат-Аврора был запущен в космос 20 июня 2014г. [88].

LEON считается перспективным процессором для электронного космического приборостроения. Он был разработан по заказу европейского космического агентства (European Space Agency, ESA) компанией Gaisler (теперь – Aeroflex Gaisler) около 10 лет назад и к настоящему времени имеет за рубежом хорошую летную историю. Он применялся на КА Colombus (2008), Proba2 (2009), JUNO (2011), SVOM/Éclair (2012), SWARM (2012), Sentinels (2013), Alphasat (2013) и многих других. Интерес к нему проявляют и отечественные разработчики космических бортовых вычислительных систем [89].

В настоящее время существуют его две версии: LEON2 и LEON3, а также соответствующие им FT версии. Оба процессора основаны на архитектуре SPARC v.8 и на архитектурном уровне по сути отличаются

только глубиной конвейера: у LEON2 он 5-ступенчатый, а у LEON3 7-ступенчатый. LEON2/ LEON2FT поддерживает ESA и европейское подразделение компании Atmel, развитием LEON3/ LEON3FT занимается по-прежнему Aeroflex Gaisler.

Процессоры доступны в виде законченных интегральных схем (AT697, AT7913, UT699 и GR712RC), а также, в частности, LEON3 доступен в виде синтезируемых IP-блоков (закрытых и открытых) для имплементирования в FPGA и ASIC [90]. Наличие открытого IP блока LEON3, специфированного на языке VHDL, предопределило его применение для создания ПМ малого космического аппарата (МКА) «ТаблетСат-Аврора», т.к. данный проект выполнялся в стесненных финансовых обстоятельствах, в короткое время, в условиях частых изменений. В качестве элементной базы для имплементирования использовалась flash-FPGA фирмы Microsemi A3PE3000L [91].

Как известно, открытая версия LEON3 не является сбоеустойчивой, поэтому в процессе проведения работ над бортовым компьютером процессор дорабатывался до сбоеустойчивой версии, способом, достаточным для миссии этого малого КА. Как и в случае закрытой сбоеустойчивой версии LEON3 FT обеспечивалась защита внутрикристальной памяти процессора (кэш-памяти и файла регистров), а также внешней памяти с помощью ECC. Для всех видов памяти использовался один и тот же отказоустойчивый код Хсяо (39,32) [92].

Код Хсяо относится к группе кодов Хемминга. Он позволяет исправлять одиночную ошибку и детектировать двойную. Применение данного кода оправдано для реализации сбоеусточивости микропроцессоров, используемых, например, для МКА с планируемым сроком активного существования не более 3-5 лет. Он малоизбыточен как к разрядности памяти, так и ресурсам FPGA, а также обладает хорошим быстродействием, что позволяет обнаруживать и исправлять ошибки «на лету».

Реализация EDAC-механизма на базе кода Хсяо соответствует функциональной схеме использования помехоустойчивого кодирования памяти, представленной на рисунке 1.2 [8]. Расчёт контрольных разрядов производится следующим образом:

CB0 = D0 ^ D4 ^ D6 ^ D7 ^ D8 ^ D9 ^ D11 ^ D14 ^ D17 ^ D18 ^ D19 ^ D21 ^ D26 ^ D28 ^ D29 ^ D31,
CB1 = D0 ^ D1 ^ D2 ^ D4 ^ D6 ^ D8 ^ D10 ^ D12 ^ D16 ^ D17 ^ D18 ^ D20 ^ D22 ^ D24 ^ D26 ^ D28,
CB2 = D0 ^ D3 ^ D4 ^ D7 ^ D9 ^ D10 ^ D13 ^ D15 ^ D16 ^ D19 ^ D20 ^ D23 ^ D25 ^ D26 ^ D29 ^ D31,
CB3 = D0 ^ D1 ^ D5 ^ D6 ^ D7 ^ D11 ^ D12 ^ D13 ^ D16 ^ D17 ^ D21 ^ D22 ^ D23 ^ D27 ^ D28 ^ D29,
CB4 = D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D14 ^ D15 ^ D18 ^ D19 ^ D20 ^ D21 ^ D22 ^ D23 ^ D30 ^ D31,
CB5 = D8 ^ D9 ^ D10 ^ D11 ^ D12 ^ D13 ^ D14 ^ D15 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31,
CB6 = D0 ^ D1 ^ D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31,

где *CBi* – значение *i* контрольного бита (*i=7*), *D(j)* – значение *j* бита исходного слова памяти (*j=32*), ^ – символ операции XOR.

Для тестирования разработанного механизма сбоеустойчивости процессора для ПМ использовался подход представленный в главе 2. Заметим, что вместо кода Хсяо может применяться любой другой, при этом инфраструктура инъекции сбоев не потребует переработки.

3.2 Детализированная структура системы инъекции сбоев для процессора LEON3

На рисунке 3.1 представлена архитектура СИС [93] для экспериментального образца ПМ малого КА «ТаблетСат-Аврора». В LEON3 OCD носит название DSU. Он является компонентом СнК и подключён к ядру процессора и ВКШ AHB AMBA. IP-блок инъектора сбоев также подключен к AHB AMBA, посредством которой и обеспечивается их взаимодействие.

Имеется два независимых канала для внесения сбоев в кэш и регистровый файл, а также во внешнюю память. Во внутреннюю память инъектор вносит сбои через DSU, а во внешнюю память через IP-блок контроллера внешней памяти MEMCTRL.

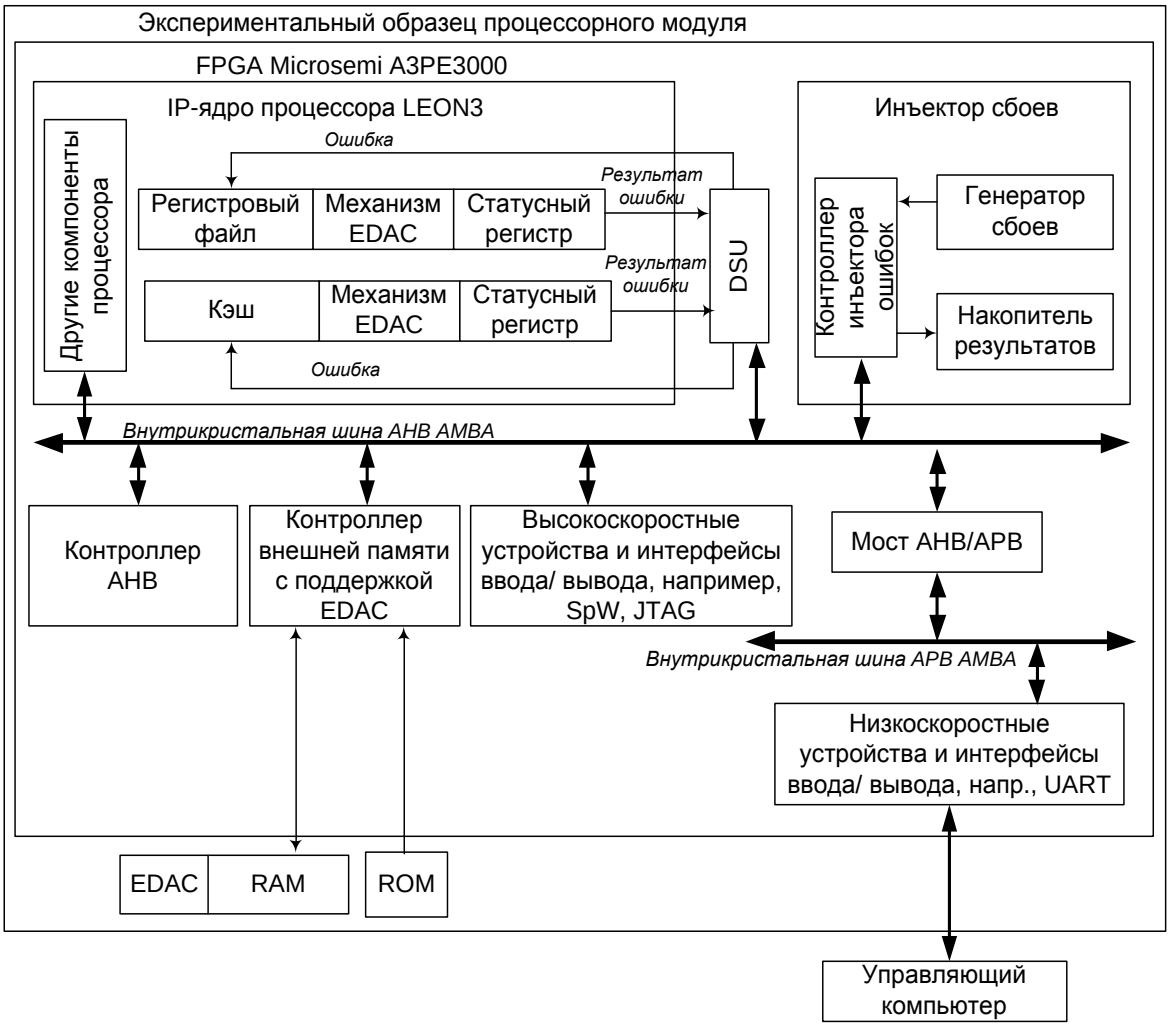


Рисунок 3.1 – Детализированная архитектура системы инъекций сбоев для LEON3

Механизм EDAC на основе кода Хсяо, дополнен специальными статусными регистрами, в которые EDAC после обнаружения и исправления ошибки заносит результаты инъекции. Если в регистре статуса EDAC появились данные об обнаруженной ошибке, контроллер FI записывает их в накопитель результатов. По окончанию FI-компании через UART ее результаты передаются в управляющий компьютер.

Первоначально отладка СИС, ее аппаратных и программных составляющих производится в модели ПМ, размещаемой в среде ModelSim 6.0 [94]. Однако FI-компании с большим количеством запланированных сбоев моделируются очень долго. Поэтому затем СИС

вместе с тестируемой моделью ПМ имплементируется в FPGA. В этом случае FI осуществляются в целевую систему, являющейся конечной целью разработки, что увеличивает значимость проводимого тестирования на сбоеустойчивость.

3.3 Аппаратное обеспечение системы инъекций сбоев

СИС является аппаратно-программной системой. В данном разделе рассмотрены некоторые ключевые аспекты разработки аппаратной части:

- структурная схема IP-блока инъектора сбоев;
- особенности функционирования составных частей инъектора сбоев в режимах инъекции с остановкой и без остановки процессора;
- особенности работы IP-блока инъектора сбоев с конвейером LEON3;
- конфигурационные настройки целевой системы, включающей IP-блок инъектора сбоев;
- модификация контроллера внешней памяти для обеспечения FI.

3.3.1 Структурная схема IP-блока инъектора сбоев

Приведенная на рисунке 3.1 структура инъектора условна и не отражает истинное устройство данного IP-блока. Она показывает только функциональный состав, который реально распределен по 6 внутренним блокам инъектора сбоев. Действительная высокоуровневая схема IP блока инъектора сбоев INJ_SEU представлена на рисунке 3.2.

IP-блок инъектора сбоев состоит из 6-ти частей (блоков). Рассмотрим их по отдельности.

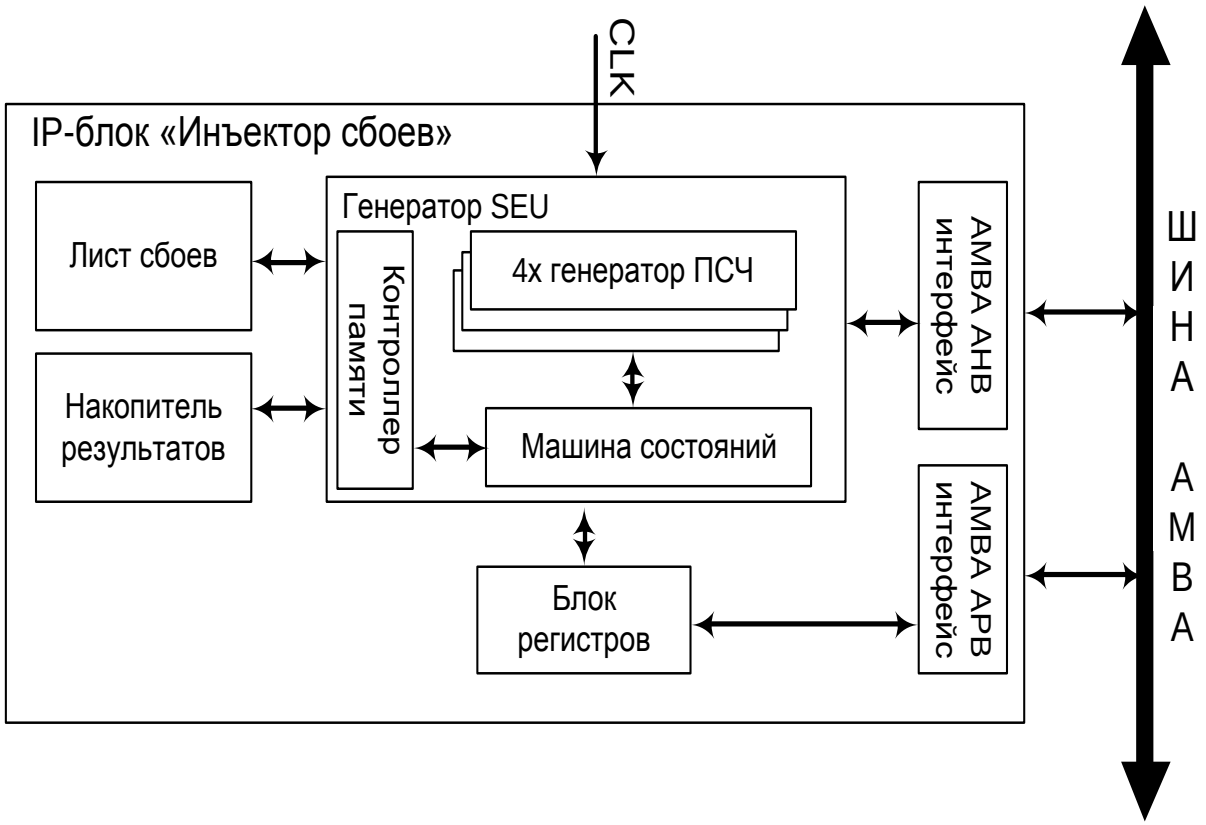


Рисунок 3.2 –Схема IP-блока инъектора сбоев.

Блок *«AMBA AHB интерфейс»* согласует инъектор сбоев с шиной AMBA AHB. При этом сам инъектор может выступать как в роли ведущего устройства на шине AMBA, так и в роли ведомого. Режим ведущего устройства позволяет инициировать транзакции по AMBA шине, необходимые для проведения инъекций сбоев в память микропроцессора. Режим ведомого устройства позволяет получить доступ к блокам «Лист сбоев» и «Накопитель результатов» другим устройствам ни шине AMBA. Данная возможность позволяет вручную инициализировать «Лист сбоев» и прочитать «Накопитель результатов», например, с помощью внешнего управляющего компьютера, подключённого к системе по UART-интерфейсу.

Запрос на инициализацию транзакции, для проведения инъекции генерируется в блоке *«Генератор SEU»*. Режим работы генератора SEU определяется соответствующей *машиной состояний* (более подробно машина

состояний описана в разделе «Описание работы инъектора сбоев»), управляемой с помощью блока регистров по AMBA APB интерфейсу.

Четыре генератора псевдослучайных чисел обеспечивают возможность автоматической генерации следующих параметров, необходимых для инъекции:

– адрес ошибки во внутренней памяти;

– положение ошибки в 32-х битом слове внутренней памяти;

– момент времени для инъекции ошибки (определяется интервалом между инъекциями, выраженным в системных тактах);

– вид ошибки: однократная (ошибка в одном бите) или двукратная (ошибка в двух соседних битах);

Генераторы псевдослучайной последовательности (ПСЧ) реализованы [95] на базе линейного регистра сдвига с обратной связью Linear Feedback Shift Register, LFSR [96]. Структура одного генератора в среде Quartus представлена на рисунке 3.3.

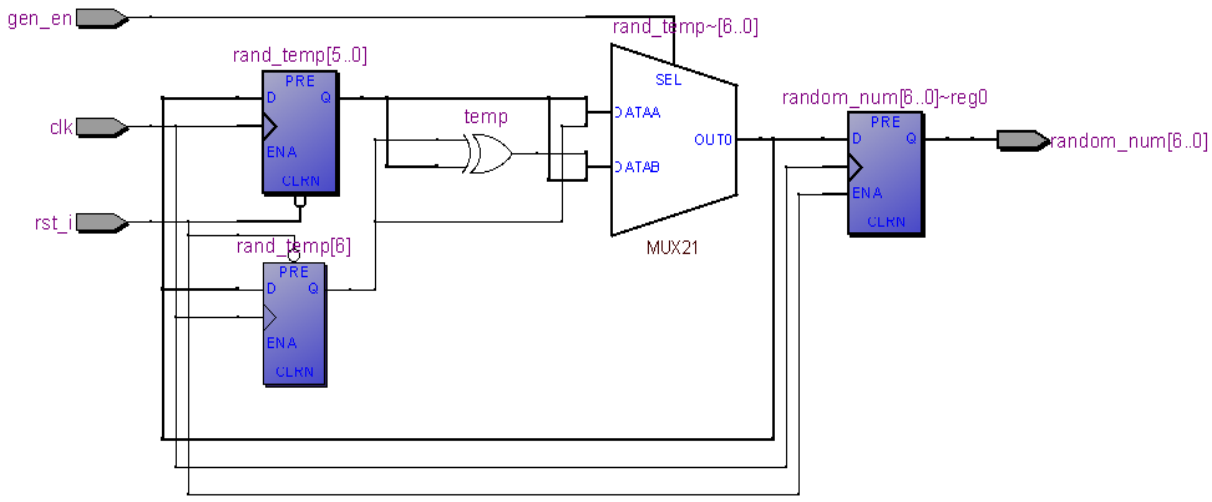


Рисунок 3.3 – Схема генератора псевдослучайной последовательности

Генерируемые данные сохраняются в блоке *«Лист сбоев»*, который представляет собой модуль внутрикристальной 32-х разрядной памяти. Для обеспечения доступа к данной памяти с различных устройств

(ручной/автоматический режимы инициализации листа сбоев) реализован контроллер памяти, обрабатывающий возможные коллизии. Следует отметить, что режим ручного редактирования возможен только при условии, что генератор сбоев не находится в активном состоянии (до начала FI-компании). Данное условие позволило сократить аппаратные ресурсы при практической реализации контроллера.

Управление инъектором сбоев осуществляется с помощью *«Блока регистров»*, доступного для чтения/записи по AMBA APB интерфейсу. Регистры блока приведены в таблице 3.1.

Таблица 3.1 – Регистры IP-ядра инъектора сбоев, располагающиеся на шине AMBA APB, длина каждого регистра 4 байта

Смещение относительно базового адреса на шине	Описание регистра
0x00	Регистр управления
0x04	Регистр задержки
0x08	Регистр статистики кэш инструкций (область тэга)
0x0c	Регистр статистики кэш инструкций (область данных)
0x10	Регистр статистики кэш данных (область тэга)
0x14	Регистр статистики кэш данных (область данных)
0x18	Регистр статистики регистровый файл
0x1C	Регистр статистики внешнего ОЗУ
0x20	Регистр предопределённого адреса
0x24	Регистр предопределённых данных

Регистр управления Control Register имеет структуру, представленную в таблице 3.2

Таблица 3.2 – Control Register

31	30:28	27:19	18	17	16	15	14	13	12	11	10	9:0
en	c_n	msb_addr	cite	cide	cdte	cdde	rfen	ren	sm	pm	rst	msbt

31 Enable (EN) – начать FI компанию;

30:28 Cycle_num (c_n) – показывает, сколько раз инъектор сбоев будет перезапущен по завершении FI-компаний. При установки данного поля в значение 0x7, генератор будет перезапускаться бесконечное количество раз;

27:19 Most significant bit address inject – биты расширения поля адреса в листе сбоев (используется для инъекций в ОЗУ)

18 Cash_i_tag_en (cite) – разрешить инъекции в кэш инструкций (область тэга);

17 Cash_i_data_en (cide) – разрешить инъекции в кэш инструкций (область данных);

16 Cash_d_tag_en (cdte) – разрешить инъекции в кэш данных (область тэга);

15 Cash_d_data_en (cdde) – разрешить инъекции в кэш данных (область данных);

14 rf_en (rfen) – разрешить инъекции в регистровый файл процессора;

13 ram_en (ren) – разрешить инъекции в ОЗУ процессора;

12 Stop Mode (SM) – включение режима работы инъектора «с остановкой процессора»;

11 Predefined Mode (PM) – включение режима работы инъектора «предопределённый»;

10 Reset (RST) – рестарт инъектора сбоев при установлении данного бита;

9:0 Most significant bit time register – биты, расширяющие Time Register.

Регистр задержки Time Register имеет структуру, представленную в таблице 3.3

Таблица 3.3 – Time Register

31		0
Time		

31:0 Time – устанавливает интервал между инъекциями (используется при равномерном инъектировании).

Регистры статистики инъектора сбоев хранят общую информацию об инъектированных/обнаруженных сбоях. Более детальная информация находится в блоке «накопитель результатов», доступ к которому осуществляется по шине AMBA AHB.

Регистр статистики кэш инструкций (область тэга) STATISTICS_CI_TAG имеет структуру, представленную в таблице 3.4

Таблица 3.4 – STATISTICS_CI_TAG

31:16	15:0
detected	Injected

31:16 Detected – показывает общее количество обнаруженных сбоев в кэше инструкций (область тэга);

15:0 Injected – показывает общее количество инъектированных сбоев в кэше инструкций (область тэга).

Регистр статистики кэш инструкций (область данных) STATISTICS_CI_DATA имеет структуру, представленную в таблице 3.5.

Таблица 3.5 – STATISTICS_CI_DATA

31:16	15:0
detected	Injected

31:16 Detected – показывает общее количество обнаруженных сбоев в кэше инструкций (область данных);

15:0 Injected – показывает общее количество инъектированных сбоев в кэше инструкций (область данных).

Регистр статистики кэш данных (область тэга) STATISTICS_CD_TAGs имеет структуру, представленную в таблице 3.6.

Таблица 3.6 – STATISTICS_CD_TAG

31:16	15:0
detected	Injected

31:16 Detected – показывает общее количество обнаруженных сбоев в кэше данных (область тэга);

15:0 Injected – показывает общее количество инъектированных сбоев в кэше данных (область тэга).

Регистр статистики кэш данных (область данных) STATISTICS_CD_DATA имеет структуру, представленную в таблице 6.4

Таблица 3.7 – STATISTICS_CD_DATA

31:16	15:0
detected	Injected

31:16 Detected – показывает общее количество обнаруженных сбоев в кэше данных (область данных);

15:0 Injected – показывает общее количество инъектированных сбоев в кэше данных (область данных).

Регистр статистики регистрового файла STATISTICS_RF имеет структуру, представленную в таблице 3.8.

Таблица 3.8 – STATISTICS_RF

31:16	15:0
detected	Injected

31:16 Detected – показывает общее количество обнаруженных сбоев в регистровом файле процессора;

15:0 Injected – показывает общее количество инъектированных сбоев в регистровом файле процессора.

Регистр статистики кэш данных (область данных) STATISTICS_RAM имеет структуру, представленную в таблице 3.9.

Таблица 3.9 – STATISTICS_RAM

31:16	15:0
detected	Injected

31:16 Detected – показывает общее количество обнаруженных сбоев в ОЗУ процессора;

15:0 Injected – показывает общее количество инъектированных сбоев в ОЗУ процессора.

Регистр предопределённого адреса PREDEFINED_ADDRESS имеет структуру, представленную в таблице 3.10.

Таблица 3.10 – PREDEFINED_ADDRESS

31 : 0
Address

31:0 – хранит значение адреса в ОЗУ, содержимое которого будет контролироваться в предопределённом режиме работы инъектора. Значение считается валидным при установленном флаге PM.

Регистр предопределённых данных PREDEFINED_DATA имеет структуру, представленную в таблице 3.11.

Таблица 3.11 – PREDEFINED_DATA

31 : 0
Address

31:0 – хранит значение которое будет сравниваться с содержимым ОЗУ (адрес указан в регистре predefined_address). При установлении данного значения контроллер начнёт производить инъекции по

данному адресу памяти. Значение считается валидным при установленном флаге PM.

«Лист сбоев» представляет собой модуль внутрикристальной памяти, разрядностью 32 бита. Структура каждой строки представлена в таблице 3.12.

Таблица 3.12 – List_SEU

31	30	29:27	26:15	14:10	9:0
en	tip	dev	error_address	error_mask	Delay

31 Enable (en) – запись актуальна (Инъектор сбоев завершает FI-компанию при прочтении неактуальной записи);

30 Ttip– определяет тип ошибки (одиночная/двойная);

29:27 Dev – определяет модуль, в который будет проводиться инъекция. "000" - регистровый файл; "001" - кэш инструкций (область тега); "010" - кэш инструкций (область данных); "011" - кэш данных (область тега); "100" - кэш данных (область данных); "101" – ОЗУ;

26:15 error_address определяет адрес инъектироваия;

14:10 error_mask определяет позицию сбоя в 32-битном слове;

9:0 delay определяет задержку между инъекциями в системных тактах. Общая задержка будет определяться по формуле: *Time Register + delay.*

«Накопитель результатов» представляет собой модуль внутрикристальной памяти, разрядностью 32 бита. Структура каждой строки представлена в таблице 3.13.

Таблица 3.13 – Statistics List

31	30	29:27	26:6	5:0
en	tip	dev	error_address	Reserved

31 Enable (en) – запись актуальна;

30 TIP – определяет тип ошибки (одиночная/двойная);

29:27 Dev – определяет модуль, в который проводилась инъекция: "000" - регистровый файл; "001" - кэш инструкций (область тега); "010" - кэш инструкций (область данных); "011" - кэш данных (область тега); "100" - кэш данных (область данных); "101" – ОЗУ;

26:6 error_address – хранит значение адреса, обнаруженной ошибки;

5:0 RESERVED – бит зарезервирован.

Как видно из структуры накопителя результатов он сохраняет адрес места в памяти, где обнаружена ошибка. Данная возможность использовалась для проверки правильности вносимых инъекций при проведении экспериментальных исследований над СИС.

Перед включением в модель целевой системы каждое IP-ядро, в том числе и INJ_SEU, настраивается соответствующим образом. В приложении А представлены настройки IP-ядра процессора LEON3, IP-ядра INJ_SEU, а также других основных IP-блоков СнК.

3.3.2 Описание работы инъектора сбоев

Работа инъектора сбоев INJ_SEU определяется машиной состояний (рисунок 3.4), входящей в состав блока «*Генератор сбоев*», управление которой определяется блоком регистров. Конечный автомат состоит из 9 состояний: idle, gen_parametrs, read_sheet, pending, snooping, stop_proc, injection, start_proc, statistics_collection. Idle – режим простоя. В этом режиме генератор сбоев не активен. У оператора есть доступ на запись для управляющих регистров и листа сбоев. Таким образом, в этом состоянии можно провести первичную инициализацию контроллера. При активации инъектора (бит «en» управляющего регистра установлен в ‘1’), генератор сбоев переходит из состояния Idle в состояние gen_parametrs или read_sheet.

Выбор определяется наличием/отсутствием флага «sh» управляющего регистра.

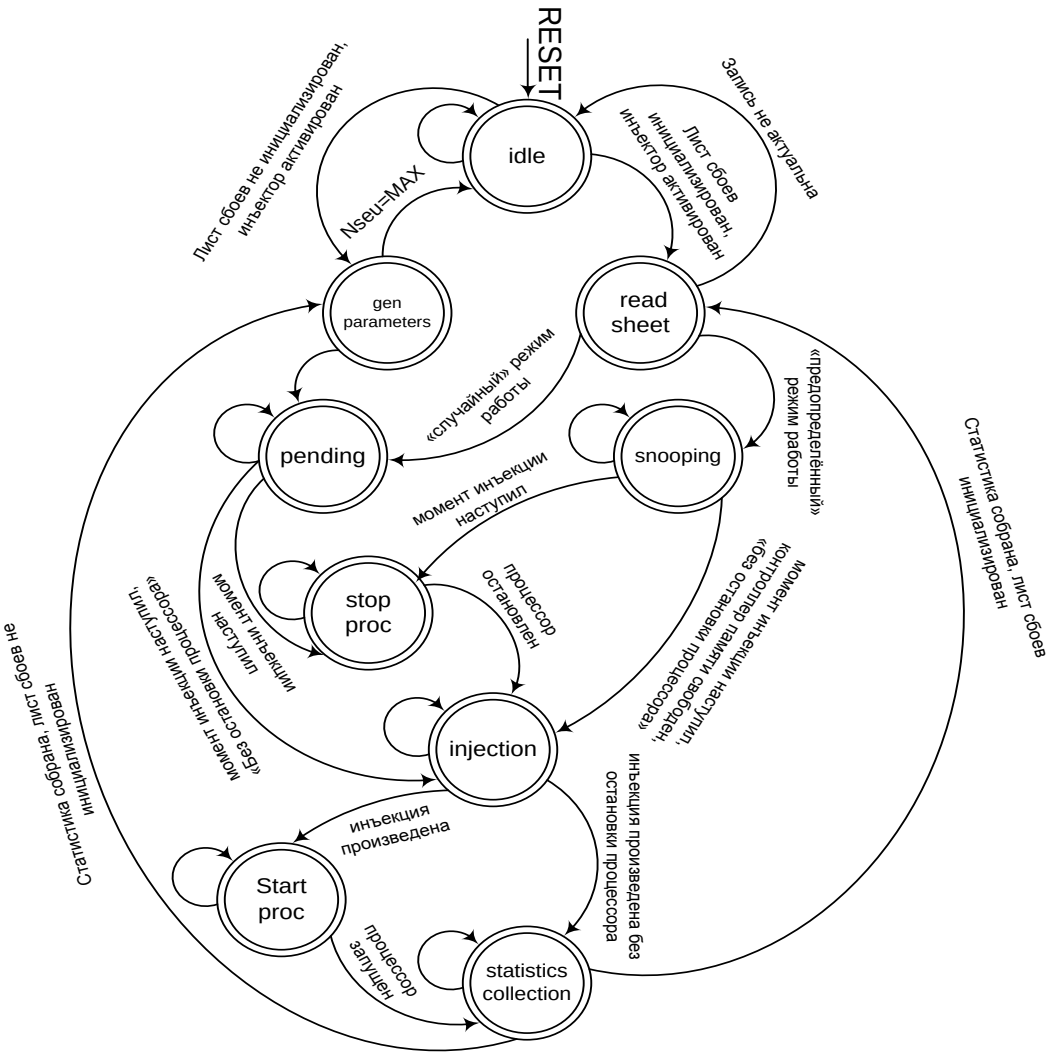


Рисунок 3.4 – Машина состояний инъектора

Если лист сбоев не был инициализирован вручную, тогда генератор сбоев сам сгенерирует необходимые параметры, использовав для этого генераторы ПСЧ в состоянии gen_parametrs. Следует отметить, что в этом случае отсутствует возможность проводить инъекции в режиме работы «предопределённый». Далее конечный автомат перейдёт в состояние «pending», ожидая момента наступления инъекции, который определяется суммой задержек, выставленной в регистре «Time» и сгенерированной

генератором ПСЧ. Затем, исходя из режима работы инъектора (с остановкой/без остановки процессора), конечный автомат перейдёт в состояние stop_proc/injection соответственно. Режим работы инъектора определяется флагом SM управляющего регистра. Следует отметить, что в режиме «без остановки процессора» инъекции возможны только в ОЗУ процессора. В состоянии injection генератор производит опирации чтения/записи данных в памяти. Адрес и положение сбоя в 32-х битном слове, в данном случае, будут определяться сгенерированными параметрами в состоянии gen_parametrs. По завершении процесса инъекции конечный автомат перейдёт либо в состояние start_proc (если активирован соответствующий режим), либо напрямую в состояние statistics_collection. Состояние statistics_collection определяет операции чтения статусных регистров механизма EDAC тестируемого процессора. По завершении процесса сбора статистики конечный автомат перейдёт в состояние gen_parametrs и FI компания будет продолжена. По завершению необходимого числа инъекций (*Nseu=max*) конечный автомат сбросит флаг «EN» в ‘0’ и вернётся в состояние Idle. Значение *max* определяется размером листа сбоев инъектора.

Если лист сбоев был инициализирован вручную, тогда конечный автомат перейдёт в состояние read_sheet с целью чтения необходимых параметров. Исходя из режима работы инъектора сбоев (определяется флагом «PM» управляющего регистра) конечный автомат перейдёт либо в состояние pending (и его работа будет соответствовать описанному выше алгоритму), либо в snooping. В состоянии snooping конечный автомат будет мониторить моменты обращения во внешнюю память процессора. При достижении необходимого watchpoint процессор будет либо остановлен в состоянии stop_proc (флаг SM управляющего регистра имеет значение ‘1’), либо при отсутствии обращений к контроллеру памяти будет произведена инъекция в ОЗУ в состоянии injection. Собрав статистику обнаруженных

ошибок в состоянии statistics_collection конечный автомат вновь перейдёт в состояние read_sheet. При достижении конца листа сбоев конечный автомат сбросит флаг «EN» в ‘0’ и вернётся в состояние Idle.

Чтение/запись данных в память процессора LEON3 производится с помощью специального отладочного интерфейса (DSU). Данный интерфейс является ведомым устройством на AMBA AHB шине. Для взаимодействия с процессором LEON3 инъектор по DMA каналу обращается к регистрам DSU. Используются набор регистров отладочного интерфейса, представленный в таблице 3.14.

Таблица 3.14 – Карта памяти DSU интерфейса, используемая инъектором сбоев для управления процессором

Смещение относительно базового адреса на шине APB	Описание регистра
0x00	DSU control register
0x000020	Break and Single Step register
0x300000 - 0x3007FC	IU register file
0x300800 - 0x300FFC	IU register file check bits (LEON3FT only)
0x400024	DSU ASI register
0x400040	ASR16
0x700000 - 0x7FFFFC	ASI diagnostic access (ASI = value in DSU ASI register, address = address[19:0])

Например, для остановки процессора LEON3 необходимо установить ‘1’ в бит Break now (BN0) регистра Break and Single Step. Для этого в состоянии stop_proc производится операция записи по адресу *DSU_start_address+0x000020* слова 0x1. Аналогично производится операция запуска процессора, путём сбрасывания данного бита в 0 в состоянии start_proc.

Для выполнения инъекций в регистровый файл необходимо отключить механизм EDAC. Для этого проводится операция записи слова 0x1 по адресу

DSU_start_address+0x400040 (регистр asr16). Далее инъектор читает/модифицирует содержимое регистрового файла, используя адрес *DSU_start_address+0x300000+error_address*. Описанные операции выполняются в состоянии injection.

Аналогично, инъектор сбоев модифицирует содержимое кэш памяти. При этом используется диагностический регистр интерфейса DSU (ASI). Инъектор по DMA каналу записывает в него (*DSU_start_address+0x400024*) следующие значения: 0x9 для доступа к кэшу инструкций (область данных), 0xB для доступа к кэшу данных (область данных), 0xC для доступа к кэшу инструкций (область тэга), 0xE для доступа к кэшу данных (область тэга). Далее производятся операции чтения/модификации по адресу *DSU_start_address+0x700000+ error_address*.

Доступ к внешнему ОЗУ осуществляется без DSU, напрямую через контроллер памяти. До осуществления инъекции инъектор сбоев отключает механизма EDAC контроллера памяти, путём установки в единицу бита RE (RAM EDAC enable) [74]. Далее осуществляется операция чтения/модификации по адресу ram_start_address+error_address.

Во всех случаях, при осуществлении инъекций модифицированные данные определяются следующим образом: $D_{mod}=D_{read}$ *xor Error_mask*, где *Error_mask* – параметр, определяемый на стадии read_sheet или gen_parametrs; D_{read} – содержимое памяти, D_{mod} – данные, записанные в результате инъекции.

Инъектор сбоев является и ведомым устройством на шине AMBA AHB. Это функция позволяет получать доступ к листу сбоев и накопителю результатов инъектора для других устройств, входящих в состав СнК. Одним из входных сигналов инъектора является *ahbsi : in ahb_slv_in_type.* Рассмотрим его состав (листинг 3.1).

```
-- AHB slave inputs
  type ahb_slv_in_type is record
    hsel  : std_logic_vector(0 to NAHBSLV-1);     -- slave select
    haddr         : std_logic_vector(31 downto 0);          -- address bus (byte)
    hwrite        : std_ulogic;                     -- read/write
    htrans        : std_logic_vector(1 downto 0);           -- transfer type
    hsize : std_logic_vector(2 downto 0);           -- transfer size
    hburst        : std_logic_vector(2 downto 0);           -- burst type
    hwdata        : std_logic_vector(31 downto 0);          -- write data bus
    hprot: std_logic_vector(3 downto 0);            -- protection control
    hready        : std_ulogic;                     -- transfer done
    hmaster       : std_logic_vector(3 downto 0);           -- current master
    hmastlock     : std_ulogic;                     -- locked access
    hmbsel        : std_logic_vector(0 to NAHBAMR-1);  -- memory bank select
    hcache        : std_ulogic;                     -- cacheable
    hirq  : std_logic_vector(NAHBIRQ-1 downto 0);       -- interrupt result bus
    testen        : std_ulogic;                     -- scan test enable
    testrst       : std_ulogic;                     -- scan test reset
    scanen        : std_ulogic;                     -- scan enable
    testoen       : std_ulogic;                     -- test output enable
  end record;
```

Листинг 3.1 – Сигнал ahbsi

Сигналы *hsel, hwrite, haddr, hmaster* позволяют инъектору контролировать шину адреса и данных, а также направление передачи (write/read). Это особенность позволяет инъектору осуществлять мониторинг транзакций на шине AMBA AHB. Данный режим используется в состоянии snooping, когда инъектор работает в привилегированном режиме и ожидает наступления определённого события (например, запись определённых данных по конкретному адресу оперативной памяти). Как только это событие происходит осуществляется инъекция.

При работе в «предопределённом режиме с остановкой процессора» инъектор может производить инъекции и в другие виды памяти. Для этого возможно установить специальные контрольные точки (watchpoint) в регистре процессора. LEON3 поддерживает до четырёх подобных контрольных точек, с возможностью наложения на них устанавливаемых масок. Таким образом, до проведения FI-компании оператор конфигурирует соответствующим образом регистры процессора и инъектор сбоев. При

срабатывании watchpoint процессор сам перейдёт в режим отладки и будет остановлен. Инъектор может определить наступление этого события по установленному биту DM (Debug mode в control register) DSU интерфейса. Ожидание наступления данного события также происходит в состоянии snooping. Далее будет произведена инъекция, путём модификации содержимого интересующей области памяти.

Таким образом, инъектор сбоев взаимодействует с памятью по DMA каналу либо через контроллер памяти, либо через DSU интерфейс. На структуру самого процессора инъектор сбоев не оказывает никакого влияния. Что делает возможным его простое включение/выключение из состава СнК. Важной особенностью инъектора является возможность одновременного внесение сбоев во все виды памяти: в регистровый файл, кэш-память и внешнюю память.

3.3.3 Модификация контроллера внешней памяти

Для реализации сбоеустойчивой внешней памяти был модифицирован открытый IP-блок контроллера памяти «MEMCTRL». Его разработчиком является ESA. В него добавлена функция расчёта контрольных разрядов при записи информационного слова во внешнюю память. Как и во всем проекте, при расчёте используется код Хсяо (39,32). При чтении информационного слова из ОЗУ контрольные разряды вновь пересчитываются. При обнаружении одиночного сбоя, информационное слово будет скорректировано и автоматически вновь записано в ОЗУ. После этого прочитанные данные будут переданы по DMA каналу запрашивающему устройству. Информация об обнаруженной ошибке будет сохранена в статусном регистре.

Состав регистров IP ядра «MEMCTRL» был изменён. За основу, ввиду обеспечения совместимости модифицированного контроллера памяти с

существующим программным обеспечением, взята версия «FTMEMCTRL» контроллера из библиотеки GRLIB. Данный IP-блок, хоть и является закрытым, но имеется описание его регистров. Основные внесенные изменения представлены ниже.

Модифицированный регистр «Memory configuration register 3» представлен в таблице 3.15.

Таблица 3.15 – Memory configuration register 3

31:28	27	26:10	9	8:0
ECNT	ME	RESERVED	RE	

31:28 ERROR COUNTER – определяет количество обнаруженных сбоев;

27 Memory EDAC (ME) – диагностирует о наличии EDAC функций у контроллера (только для чтения);

11:10 RESERVED;

9 RAM EDAC enable (RE) – разрешить работу EDAC механизма;

8:0 RESERVED.

Таким образом, в отладочных целях можно отключить работу EDAC механизма, модифицировать содержимое внешней памяти и вновь включить работу EDAC.

Для хранения адреса последнего обнаруженного сбоя был добавлен регистр «ECC_ADDR_STAT», представленный в таблице 3.16.

Таблица 3.16 – ECC_ADDR_STAT

31:0
Memory address

31:0 Memory address – адрес последнего обнаруженного сбоя.

Синтез модифицированного IP ядра контроллера памяти проводился для ПЛИС A3PE3000L. В результате модифицированный контроллер памяти

занимает 983 логических элемента против 677 элементов у стандартного ядра.

Функциональные изменения, внесённые в IP-блок «MEMCTRL», являются незначительными и связаны только с добавлением механизма EDAC. Более подробно с особенностями проведенной модернизации IP-блока «MEMCTRL» можно познакомиться в [97].

3.3.4 Работа конвейера LEON3 при обнаружении сбоя в регистровом файле

Важной задачей при реализации EDAC-механизма регистрового файла на основе кода Хсяо является модификация работы конвейера LEON3. Конвейер имеет 7 стадий (таблица 3.17). При выполнении инструкции instr2 в момент чтения данных из регистрового файла (стадия Register Access) механизм ECC обнаруживает сбой путём сравнения контрольных разрядов 39-битного слова с ожидаемыми значениями (Check). Если ошибка однократная, она автоматически исправляется и исправленное слово (Corr) остается доступным процессору на остальных стадиях работы конвейера. Однако, вместо того чтобы на стадии Write-back записать в регистры процессора результат текущей инструкции необходимо переписать содержимое регистра на исправленное значение, а затем заново начать выполнять instr2 путем перезапуска конвейера. Иначе ошибочное значение остается в регистре и при новом сбое в нем ошибка уже станет двойной, которую код Хсяо уже не сможет исправить. Выполнить это можно только на стадии Write-back: одновременно происходит сброс конвейера (Flush) и перезапись содержимого регистра (update). Как следует из таблицы, задержка на перезапуск конвейера составит 6 тактов (instr2 вновь будет на стадии Register Access). Таким образом, коррекция одиночных сбоев происходит прозрачно для ПО.

Таблица 3.17 – Работа конвейера при обнаружении одиночного сбоя

Стадия	Инструкция											
Fetch	instr1	instr2	instr3	instr4	instr5	instr6	instr7	FLUSH	instr2	instr3	instr4	…
Decode		instr1	instr2	instr3	instr4	instr5	instr6	FLUSH		instr2	instr3	…
Register Access			instr1	check	instr3	instr4	instr5	FLUSH			instr2	…
Execute				instr1	corr	instr3	instr4	FLUSH				…
Memory					instr1	corr	instr3	FLUSH				…
Exception						instr1	corr	FLUSH				…
Write-back							instr1	update				…

При обнаружении двойной ошибки, которую с помощью кода Хсяо (39,32) исправить невозможно, процессор выдаёт соответствующее исключение (TRAP), которое затем обрабатывается ПО (таблица 3.18). Таким образом, с помощью данного метода можно тестировать реакцию бортового на двойные сбои в памяти.

Таблица 3.18 – Работа конвейера при обнаружении двойного сбоя

Стадия	Инструкция									
Fetch	instr1	instr2	instr3	instr4	instr5	instr6	instr7	FLUSH	TRAP1	TRAP2
Decode		instr1	instr2	instr3	instr4	instr5	instr6	FLUSH		TRAP1
Register Access			instr1	check	instr3	instr4	instr5	FLUSH		
Execute				instr1	error	instr3	instr4	FLUSH		
Memory					instr1	error	instr3	FLUSH		
Exception						instr1	error	FLUSH		
Write-back							instr1	TRAP		

Для внесения изменений в работу конвейера был модифицирован алгоритм работы АЛУ процессора LEON3. Поясним, что данное изменение не относится к СИС, оно производится в ядре процессора, и оно неизбежно при создании его сбоеустойчивой версии. Изменения определяются видом

ECC. Для кодов с исправлением одной ошибки, как у кода Хсяо, работа с конвейером и соответствующие изменения в ядре процессора будут одинаковы.

3.4 Программное обеспечение системы инъекций сбоев

Программное обеспечение СИС состоит из тестовой программы, поддерживающей выполнение того или иного сценария тестирования, ту или иную FI-компанию. ПО СИС можно квалифицировать как программный комплекс, состоящий из некоторого количества тестов, объединенных под одной программной оболочкой из одного исполняемого файла.

Файл программы называется SEU_TEST. Программа выполняется в целевой системе, т.е. в тестируемом процессоре ПМ, причем как в модели процессора, так и в процессоре, имплементированном в FPGA.

Программа написана на СИ. Среда разработки, компилятор – sparc-elf-gcc v.3.4.4 (Aeroflex Gaisler). Программа не требует операционной системы. Для загрузки программы в LEON3 необходим загрузчик MKPROM2 (Aeroflex Gaisler). Размер файла программы 736Кбайт.

Используемая технология тестирования на сбоеустойчивость соответствует общей методологии тестирования СнК [98, 99]. Для создания тестовых программных воздействий на сложно-функциональные блоки используется собственный процессорный IP-блок. Тестовое ПО подготавливается на внешнем компьютере, а затем загружается в память СнК и исполняется с помощью его процессорного ядра. Эта особенность расширяет функциональные возможности тестирования, так как тестовые воздействия генерируются внутри системы и имеют доступ ко всем узлам системы и их функциям, к которым нет доступа извне. Причем разработанные программные тестовые воздействия вначале используют для тестирования модели, а затем без изменения используются для тестирования законченного устройства.

Программа тестирования SEU_TEST обеспечивает выполнение следующих тестов:

– тест на инъекции одиночных сбоев в регистровый файл процессора с остановкой процессора со случайным покрытием сбоями;

– тест на инъекции одиночных сбоев в кэш инструкций (область тэга) процессора с остановкой процессора со случайным покрытием сбоями;

– тест на инъекции одиночных сбоев в кэш инструкций (область данных) процессора с остановкой процессора со случайным покрытием сбоями;

– тест на инъекции одиночных сбоев в кэш данных (область тэга) процессора с остановкой процессора со случайным покрытием сбоями;

– тест на инъекции одиночных сбоев в кэш данных (область данных) процессора с остановкой процессора со случайным покрытием сбоями;

– тест на инъекции одиночных сбоев во внешнюю память с остановкой процессора со случайным покрытием сбоями;

– тест сбоеустойчивости при смешанном режиме инъекций сбоев в память (инъекции вводятся как в ОЗУ, так и во внутрикристальную память процессора) с остановкой процессора со случайным покрытием сбоями;

– тест на инъекции одиночных сбоев в регистровый файл процессора с остановкой процессора с предопределенным покрытием сбоями;

– тест на инъекции одиночных сбоев во внешнюю память без остановки процессора со случайным покрытием сбоями;

– тест на инъекции одиночных сбоев во внешнюю память без остановки процессора с предопределенным покрытием сбоями.

– тест на инъекции двойных сбоев в регистровый файл процессора с остановкой процессора со случайным покрытием сбоями;

– тест на инъекции двойных сбоев в кэш данных (область данных) процессора с остановкой процессора со случайным покрытием сбоями.

Некоторые тесты позволяют отработать некоторые дополнительные варианты улучшения сбоеустойчивости целевого ПО. Например, в тесте на

инъектирование двойной ошибки в кэш-данных при обнаружении двойной ошибки данные будут признаны не валидными. Будет сымитирована ситуация отсутствия попадания данных в кэш. После этого процессор заменит содержимое кэш из внешнего ОЗУ и продолжит выполнение нагрузочной программы. В качестве нагрузочной программы может быть использована любая программа.

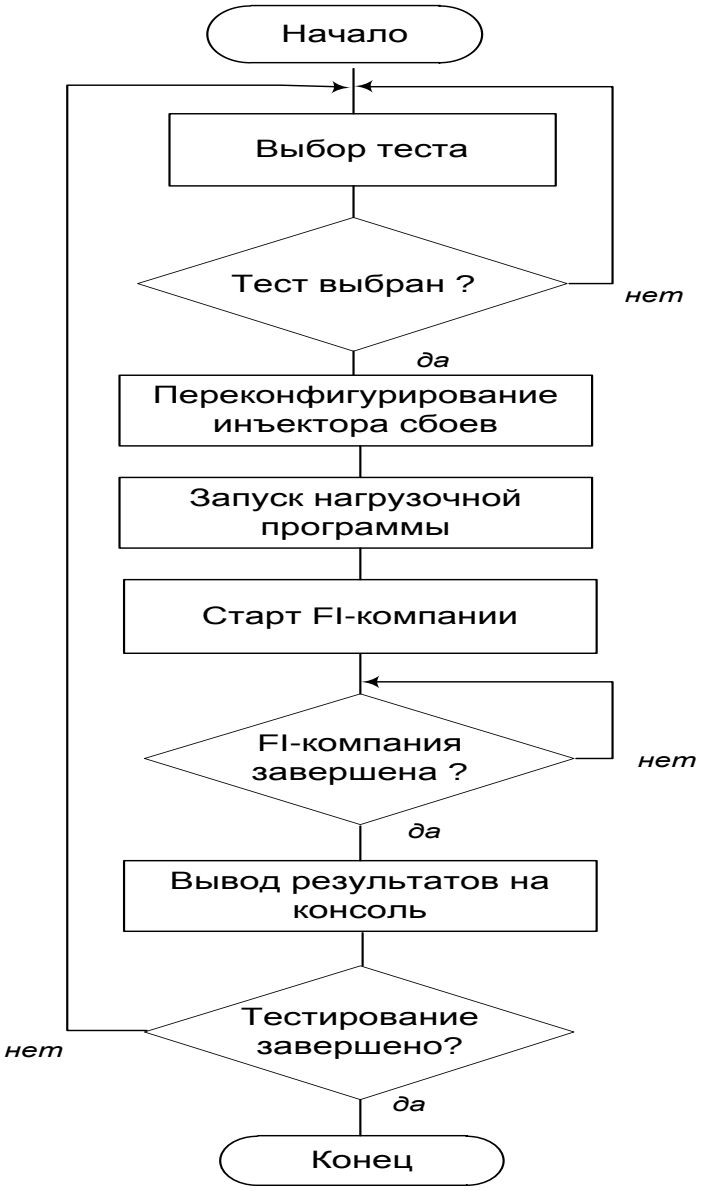


Рисунок 3.5 – Общий алгоритм выполнения программы SEU_TEST

Тесты со случайным покрытием имеют два варианта формирования листа сбоев: или сформированного и предварительно загруженного из управляющего компьютера или автоматически полученного с помощью генератора псевдослучайных чисел инъектора сбоев.

Тесты также задавались на разное количество инъекций и разную частоту сбоев в минуту. Однако, из-за ограничений консольного режима в целевой системе возможность задания данных параметров не была автоматизирована в процессе выполнения тестовой программы и данные параметры сразу задавались при подготовке указанных выше тестов перед проведением запланированной FI-компании. Данный режим был признан как адекватный исследовательскому характеру проводимой работы.

Общий алгоритм работы SEU_TEST представлен на рисунке 3.5. При вызове программы SEU_TEST появляется запрос на выбор конкретного теста. Вид экрана для выбора на рисунке 3.6. Выбор тестового воздействия определяется выбором соответствующей подпрограммы по его номеру в консольном режиме. Если тест выбран, то происходит переконфигурирование инъектора сбоев с помощью установок в статусные регистры инъектора, запуск фоновой нагрузочной программы, и старт работы выбранной FI-компании. Далее происходят аппаратные FI, а программа просто ожидает их окончания.

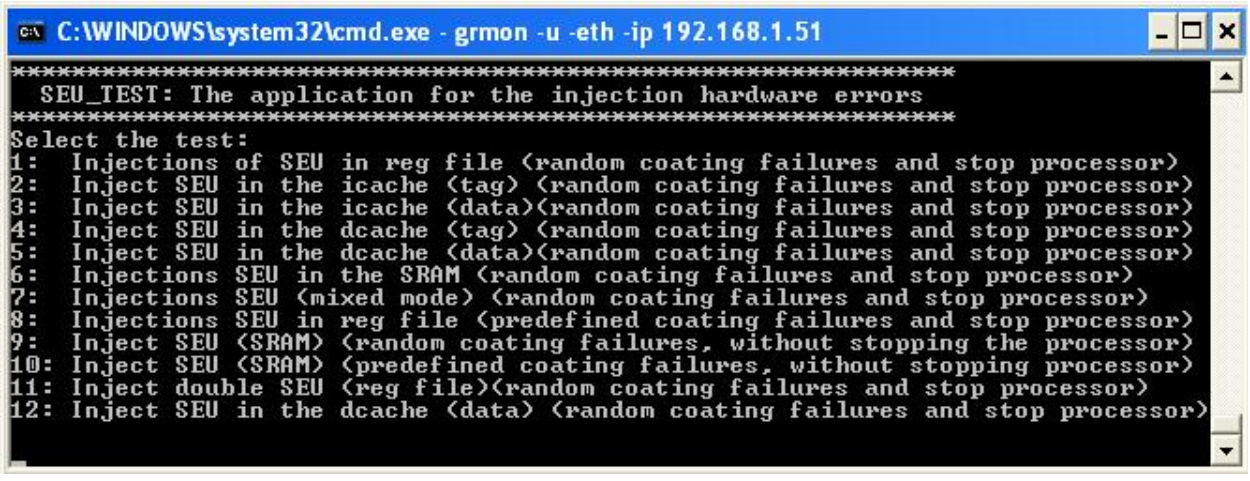


Рисунок 3.6 – Запрос на выбор теста

При успешном выполнении теста на экране появится соответствующее сообщение, сопровождаемое дополнительной информацией об окончании нагрузочной программы и результатах тестирования (рисунок 3.7). По окончанию исполнения теста появится запрос на продолжение тестирования. Если выбирается продолжение тестирования, то происходит переход к началу алгоритма и появится запрос о выборе программного теста.

Фоновой программой, выполняемой в процессе инъекции сбоев, в зависимости от целей тестирования, режима работы процессора и способом покрытия памяти сбоями могут быть:

- тест производительности процессора «Single Precision C/C++ Whetstone Benchmark» [100], оценивающий производительность процессора Данный тест использовался в режиме с остановкой процессора со случайным покрытием сбоями внутренней памяти;

- программа «RTEMS-tasks», входящая в состав примеров программ для процессора LEON3, выполняемых под управлением ОС RTEMS [101]. Данная программа демонстрирует базовые возможности переключения между задачами в ОС RTEMS. В программе генерируются три задачи, выполняемые параллельно. Каждая из них опрашивает состояние системного таймера и вычисляет текущее системное время в формате <час:мин:сек месяц:день:год>. Значение системного времени выводится пользователю на консоль. Данная программа использовалась в режиме с остановкой процессора со случайным покрытием сбоями внутренней памяти;

- специально написанная программа, в которой выделялся и инициализировался массив данных, равный размеру ОЗУ. Далее в цикле происходил расчет контрольной суммы данных, расположенных в массиве. Если механизм ECC отрабатывал вносимые ошибки, то контрольная сумма после каждой итерации должна оставаться одной и той же. Программа использовалась как в режиме с остановкой процессора, так и без остановки процессора со случайным покрытием сбоями внутренней памяти;

- специально написанная программа «Hardware monitor». Данная программа осуществляет анализ показаний датчиков тока и температуры процессорного модуля. При принятии команды запроса телеметрии программа собирает текущие показания датчиков в SpaceWire пакет и отправляет их по сети SpaceWire. Поле обратного адреса берётся из принятой команды. Кроме того, при достижении критических значений с датчиков тока и температуры «Hardware monitor» обязан самостоятельно сгенерировать пакет телеметрической информации и отправить его в сеть. Инъектор сбоев был сконфигурирован на генерацию инъекций в следующие области памяти: область хранения показаний датчиков тока и температуры; область хранения принятых командных пакетов по сети SpaceWire. Инъектор контролировал момент записи SpaceWire пакета в ОЗУ и модифицировал поле обратного адреса, что могло привести к потере телеметрической информации. Также вносились искажения в записываемые показания датчиков тока и температуры, что могло привести к ошибочной отправке пакетов с критической телеметрической информацией.

- несколько простых программ работы с большими массивам, инициализированными известными данными, в которых выполнялись простые операции по последовательному подсчету всех элементов по строкам и столбцам. Программы использовались в режиме без остановки процессора с предопределенным покрытием сбоями, а также для предопределенного занесения в регистровый файл с остановкой процессора. Сбои инъектировались при достижении определенных переменных, размещенных как во внешней памяти, так и в регистрах процессора, заранее известных значений, например при достижения счетчика цикла предельного значения, а также при переходах по условным операторам. Инженер тестирования может самостоятельно разработать нагрузочную программу, соответствующую целям планируемого тестирования на сбоеустойчивость.

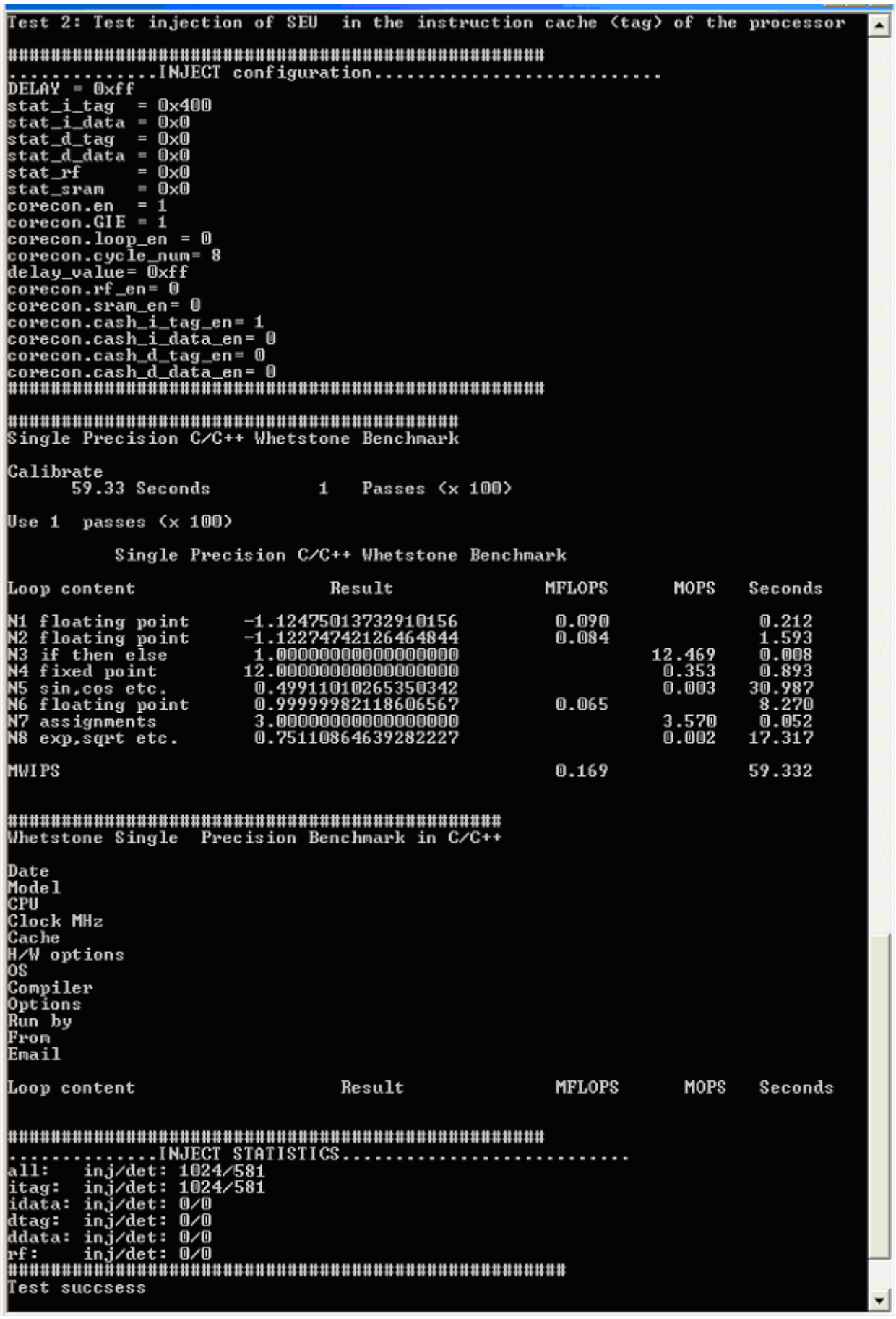


Рисунок 3.7 – Результат теста инъекции сбоев в кэш-данных в область тэга.

Результаты тестирования в различных режимах и способах покрытия памяти сбоями, а также их анализ приведены в главе 4.

В рамках соглашения с Минобрнауки о предоставлении субсидии от «19» июня 2015 г. № 14.574.21.0041 ФЦП «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса на 2014-2020 годы» по теме «Разработка бортового комплекса управления на базе технологии система на кристалле для цифровой платформы сверхмалого космического аппарата» № госрегистрации 114090470021 разработана программная документация «Прикладная программа инъекций аппаратных ошибок» на 85 листах, состоящая из документов, определенных в спецификации RU.СГАУ.56347 01: текст программы, описание программы, описание применения программы, руководство системного программиста.
Получено Свидетельство о государственной регистрации программы для ЭВМ «Программа для тестирования процессоров с помощью инъекции одиночных сбоев во внутреннюю память» № 2015614979 от 05.05.2015 [102]. В приложении Б приведена его копия.

3.5 Выводы по главе

Результаты проведенных исследований позволяют сделать следующие выводы:

1. Разработанный инъектор сбоев является независимым от ядра процессора сложно-функциональным блоком, позволяющим автономно проводить сбои как во внутреннюю, так и во внешнюю память целевой системы.

2. Разработанное тестовое программное обеспечение позволяет осуществлять инъекций сбоев в режимах с остановкой и без остановки процессора со случайным и предопределенным покрытием как внутренней, так и внешней памяти.

3. Проведенная разработка подтвердила реализуемость предложенного подхода инъектирования сбоев в микропроцессор типа система на кристалле с помощью специального аппаратного блока для инъекций, с использованием встроенного внутрикристального отладчика.

ГЛАВА 4 РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТАЛЬНЫХ ИССЛЕДОВАНИЙ

В **четвертой главе** приведены результаты экспериментальных исследований эффективности предложенных решений для тестирования сбоеустойчивости процессора LEON3.

4.1 Планирование экспериментальных исследований

Экспериментальные исследования заключались в проведении двух видов тестирования: верификации самой СИС и демонстрации возможностей разработанного метода инъекций сбоев. Перечень экспериментальных исследований представлен в таблице 4.1.

Таблица 4.1 – Виды экспериментальных исследований

Группа	Вид исследования
Верификация СИС	1. Тестирование реализации кода Хсяо 2. Тестирование правильности инъекций 3. Оценка задержек инъектирования 4. Оценка затратности ресурсов FPGA
Демонстрация возможностей СИС	1. Тестирование длительности FI-экспериментов с остановкой процессора 2. Инъекции из предопределенного листа событий (одинарных/двойных) с остановкой процессора 3. Инъекции с разной скоростью сбоев (одинарных/двойных) для режима с остановкой 4. Тестирование со случайным равномерным распределением сбоев (одинарных/двойных) с остановкой процессора из автоматически генерируемого листа событий 5. Тест при смешанном режиме инъекций сбоев в память (инъекции вводятся как в ОЗУ, так и во внутрикристальную память процессора) с остановкой процессора со случайным покрытием сбоями 6. Тест на инъекции одиночных сбоев в регистровый файл процессора с

Группа	Вид исследования
	остановкой процессора с предопределенным покрытием сбоями 7. Тест на инъекции одиночных сбоев во внешнюю память без остановки процессора с предопределенным покрытием сбоями

Верификация СИС проводится с целью удостоверения, что реализованные решения действительно выполняются правильно. Демонстрация возможностей показывает варианты применения разработанной СИС для различных случаев. Она использует программу тестирования SEU_TEST, рассчитанную на все возможные режимы работы инъектора, при этом для режима с остановкой процессора и случайным покрытием внутренней памяти варьируются максимальное количество вносимых сбоев и скорость инъекций.

4.2 Верификация системы для инъекции сбоев

Верификация разработанной СИС производилась с целью исследования достоверности разработанных решений, оценки задержек и затрат ресурсов.

Тестирование реализации кода Хсяо осуществлялось с помощью теста для программной инъекции сбоев из библиотеки GRLIB компании AeroFlex Gaisler [74], являющейся разработчикам СнК-процессора LEON3. Программа имеет название leon3_test. Она производит тестирование следующих составляющих процессора: регистровый файл, аппаратный умножитель, аппаратный делитель, кэш память, блок управления памятью (MMU) в соответсвии со схемой инъектирования, представленной на рисунке 1.10. При тестировании регистрового файла осуществляются программные инъекции сбоев, осуществляемые перед обращением к регистровой памяти с операцией чтения. Для целей испытания регистрового файла на корректность работы EDAC механизма контрольные биты регистровой памяти изменяются программно. Это делается путем установки ITE бита регистра asr16 в ‘1’. В

этом режиме, прежде чем происходит операция записи в регистровый файл, проводится операция XOR рассчитанных контрольных битов с содержанием регистра %asr16.FTB. При чтении записанного слова проверяются статусы счётчиков ошибок регистрового файла. Они должны быть инкрементированы. Результаты выполнения теста представлены на рисунке 4.1.

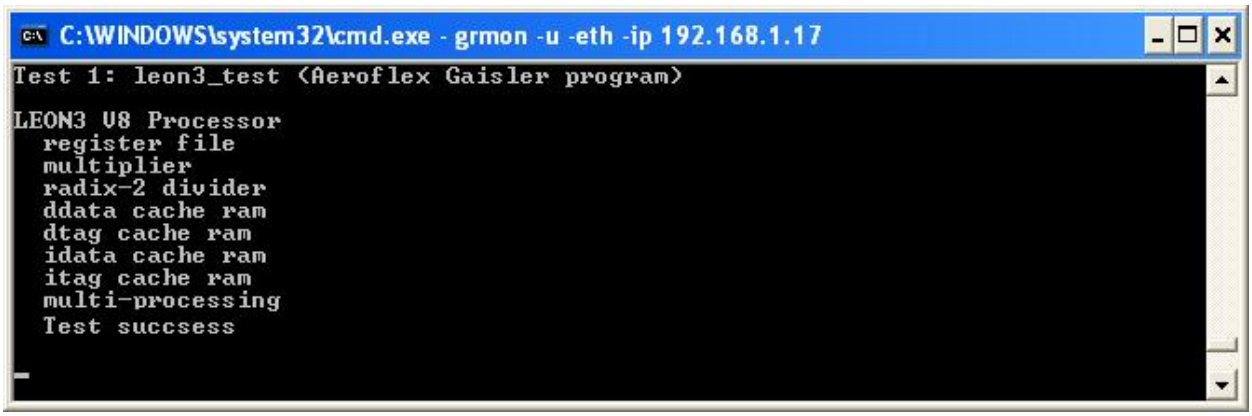


Рисунок 4.1 – Результат программной инъекции сбоев в регистровый файл и кэш память

Аналогично происходят инъекции в кэш память. В качестве маски для контрольных разрядов кэш памяти используются TB биты Cache Control Register процессора LEON3. Выполнение стандартных тестов процессора, разработанных компанией AeroFlex Gaisler с одной стороны подтвердило правильность работы разработанного EDAC-механизма внутрикристальной памяти на основе кода Хсяо, с другой показало функциональное соответствие модифицированного IP ядра процессора LEON3 его закрытой версии.

Правильность производимых инъекций осуществлялась сравнением местоположения эффективных ошибок с исходным местоположением сбоев, которые были заранее определены в листе событий. Инъектор сбоев был настроен на внесение инъекций во все доступные виды памяти с остановкой процессора. Размер листа сбоев составлял 512 байт, то есть 128 строк по 32 бита в каждой строке. В качестве нагрузочного ПО была использована

программа «Single Precision C/C++ Whetstone Benchmark» [100], оценивающая производительность процессора. Всего было инъектировано 128 сбоев, 42 из них оказались эффективными, т.е. приведшими к срабатыванию механизма EDAC. В таблице 4.2 приведены результаты тестирования, собранные в накопителе результатов и представленные в удобной форме.

Таблица 4.2 – Содержимое накопителя результатов

Тип памяти	Тип ошибки	Адрес в памяти
кэш данных (область данных)	одиночная	0x2f6
кэш инструкций (область данных)	одиночная	0xa4
SRAM-память	одиночная	0x3a0
кэш данных (область тега)	одиночная	0x78
регистровый файл	одиночная	0x1ae
кэш инструкций (область тега)	одиночная	0x1f8
SRAM-память	одиночная	0x2bc
кэш инструкций (область данных)	одиночная	0x50
SRAM-память	одиночная	0x20c
SRAM-память	одиночная	0x290
кэш инструкций (область тега)	одиночная	0xd8
кэш инструкций (область данных)	одиночная	0x144
регистровый файл	одиночная	0x1cc
SRAM-память	одиночная	0xec
кэш данных (область данных)	одиночная	0xb0
кэш данных (область тега)	одиночная	0x144
кэш данных (область тега)	одиночная	0x150
кэш инструкций (область данных)	одиночная	0xc4
регистровый файл	одиночная	0x98
кэш инструкций (область тега)	одиночная	0x28c
кэш инструкций (область тега)	одиночная	0x1c
кэш инструкций (область данных)	одиночная	0x38
кэш инструкций (область данных)	одиночная	0xdc
кэш инструкций (область тега)	одиночная	0x190
кэш данных (область данных)	двойная	0x164

Тип памяти	Тип ошибки	Адрес в памяти
кэш данных (область тега)	одиночная	0x204
кэш инструкций (область тега)	одиночная	0x274
регистровый файл	одиночная	0x5c
кэш данных (область данных)	одиночная	0x380
SRAM-память	одиночная	0x1a8
кэш инструкций (область тега)	одиночная	0x210
кэш инструкций (область тега)	одиночная	0x1c
кэш инструкций (область данных)	двойная	0xcc
регистровый файл	одиночная	0x40
кэш данных (область данных)	одиночная	0x130
кэш инструкций (область тега)	одиночная	0x28
регистровый файл	одиночная	0x50
SRAM-память	одиночная	0x304
кэш данных (область тега)	одиночная	0xac
кэш инструкций (область тега)	одиночная	0xd8
кэш инструкций (область тега)	одиночная	0x10
SRAM-память	одиночная	0x1a8

Все адреса эффективных событий совпали с адресами из листа сбоев, что свидетельствует о корректности работы IP блока инъектора сбоев.

Анализ процессов, происходящих в процессоре LEON3 при инъекции одного сбоя в режиме с остановкой процессора со случайным покрытием, показал, что *задержку* от момента генерации очередной ошибки до записи ее во внутреннюю память $t_з$ можно определить по выражению

$$t_з = \frac{N_0 \pm \Delta_N}{F}, \qquad (4.1)$$

где: $N_0 = 54$ – среднее число тактов, необходимых для внесения инъекции (получено на основании поведенческой модели процессора), $\Delta_N = 7$ – максимальное отклонение от среднего показателя числа тактов, необходимых для внесения инъекций; F – системная частота.

При F = 25 МГц задержка внесения инъекции находится в диапазоне 1,7 - 2,4 мкс. Плавающее значение задержки обусловлено особенностями работы

контроллера внутрикристальной шины AMBA 2.0. При увеличении тактовой частоты, задержка будет пропорционально уменьшаться.

Для оценки избыточности представленная СИС была реализована для разного количества ошибок, предварительно определенных в листе сбоев, и для нескольких FPGA компаний Microsemi и ALTERA (таблица 4.3). В скобках указан процент от общего количества ресурса.

Таблица 4.3 – Результаты синтеза инъектора сбоев в FPGA

Число инъекций	Тип FPGA					
	Microsemi A3PE3000L		Microsemi RTAX 1000S		ALTERA Cyclone IV	
	Логические элементы	Внутри-кристальная память	Логические элементы	Внутри-кристальная память	Логические элементы	Внутри-кристальная память
100	1285 (1.7%)	2 (1.7%)	301 (1.5%)	2 (5%)	370 (<1%)	2.176 (<1%)
500	1301 (1.7%)	3 (2.6%)	386 (2.1%)	3 (8%)	369 (<1%)	8 (<1%)
2000	1337 (1.7%)	9 (8.0%)	448 (2.4%)	9 (25%)	379 (<1%)	34 (<1%)

Результаты, представленные в таблице 4.3, показывают, что предложенный подход не является ресурсозатратным. С ростом размера листа ошибок количество логических блоков, необходимых для реализации метода, остается практически неизменным и не превышает 2%; размер внутрикристальной памяти увеличивается пропорционально. Применение в предложенном методе функции автоматической генерации сбоев позволяет уменьшить объем требуемой внутрикристальной памяти FPGA, так как для работы генератора сбоев в автоматическом режиме нет необходимости хранить большие объемы данных.

4.3 Демонстрация возможностей системы для инъекций сбоев

Для демонстрации возможностей метода было разработано множество FI-компаний экспериментов. Ниже представлены их описания и результаты.

В таблице 4.4 представлены результаты инъекций в регистровый файл арифметико-логического устройства процессора. Генерация сбоев проводилась автоматически. Генератор сбоев был настроен на внесение инъекций по случайному равномерному закону распределения. В течение теста процессор выполнял тестовую программу, которая в цикле последовательно обращалась к регистрам с операциями чтения/записи таким образом, чтобы увеличить процент эффективных сбоев. Инъектор сбоев работал в режиме с остановкой процессора.

Таблица 4.4 – Результаты инъекций в регистровый файл

Число инъекций	Число исправленных сбоев в регистровом файле		
	интенсивность 1 событие в минуту	интенсивность 10 событий в минуту	интенсивность 100 событий в минуту
100	45 (45%)	51 (51%)	58 (58%)
500	263 (52,6%)	279 (55.8%)	218 (43.6%)
1000	581 (58,1)%	613 (61.3%)	641 (64.1%)
2000	1118 (55,9%)	1097 (54.8%)	1358 (67.9%)

Все одиночные сбои, обнаруженные в регистровом файле были парированы и не привели к катастрофическим отказам, т.е. частота катастрофических отказов v=0. Подобный результат наблюдался при различных значениях v_{seu}: 1, 10, 100 событий в минуту. В реальных условиях эксплуатации в пределах околоземного космического пространства v_{seu} значительно ниже 100.

В таблице 4.5 представлены результаты инъекций в кэш процессора, которые также генерировались автоматически случайным образом с остановкой процессора. Предельная скорость сбоев v_{seu} при проведении

эксперимента составляла 100 событий в минуту. При этом в качестве тестовой использовалась программа «rtems-tasks», входящая в состав примеров программ для процессора LEON3, выполняемых под управлением ОС RTEMS. Данная программа демонстрирует базовые возможности переключения между задачами в ОС RTEMS. В программе генерируются три задачи, выполняемые параллельно. Каждая из них опрашивает состояние системного таймера и вычисляет текущее системное время в формате <час:мин:сек месяц:день:год>. Значение системного времени выводится пользователю на консоль. Успешное функционирование данной программы в течение длительного времени под воздействием одиночных инъекций в кэш-память процессора подтвердило правильность работы тестируемых функций сбоеустойчивости.

Таблица 4.5 – Результаты инъекций в кэш-память

Число инъекций	Число исправленных сбоев в кэш			
	кэш команд		кэш данных	
	tag	data	tag	Data
100	62 (62.0%)	57 (57.0%)	48 (48.0%)	54.0 (54%)
500	279 (55.8%)	218 (43.6%)	261 (52.2%)	227 (45.4%)
1000	529 (52.9%)	513 (51.3%)	678 (67.8%)	612 (61.2%)
2000	1283 (64.5%)	1336 (66.8%)	1227 (61.3%)	1340 (67.0%)

Как следует из таблиц 4.4 и 4.5 максимальное число произведенных инъекций равно 2000. И это не предельная величина – она может быть существенно выше, поэтому разработанная СИС позволяет проводить статистически значимые FI-компании для получения достоверной оценки критерия покрытия сбоев.

В таблице 4.6 представлены результаты инъекций при проведении инъекций из предопределенного листа сбоев (одинарных/двойных) с остановкой процессора. Сбои инъектировались во все виды памяти (ОЗУ, кэш память, регистровый файл). В листе сбоев содержалось 128 записей об

инъекциях. Инъектор был сконфигурирован на проведение 4-х FI компаний подряд. Таким образом, общее количество сбоев, по завершению 4-х компаний, составляло 512 сбоев. В качестве нагрузочного ПО была использована программа «Single Precision C/C++ Whetstone Benchmark», оценивающая производительность процессора. При выполнении данной программы энергопотребление процессора резко возрастает. Это связано с частым обращением к различным видам памяти при выполнении сложных арифметических операций, заданных в данной программе.

Не все сбои оказались эффективными, однако все одиночные сбои были исправлены. Двойные ошибки (их количество показано в скобках), обнаруженные в SRAM памяти и регистровом файле, привели к переводу процессора в режим остановки. Затем он был перезагружен по аппаратному WatchDog таймеру. Тестовое программное обеспечение, хранящееся во flash памяти, было вновь размещено в ОЗУ программным универсальным загрузчиком u-boot и заново запустилось на исполнение. Инъектор не имеет сброса по WatchDog, поэтому его работа не прекращалась.

Таблица 4.6 – Результаты инъекций из предопределённого листа событий.

Номер FI компании	Число одиночных (двойных) сбоев в памяти					
	кэш команд		кэш данных		Регистровый файл	SRAM память
	tag	data	tag	data		
1-я компания	10 (1)	7	5	4 (1)	6	8
2-я компания	10	8 (2)	2	7 (1)	4	4
3-я компания	7	3	6	13	6(1)	12
4-я компания	8	12	7	10	3	6(1)

В таблице 4.7 приведены результаты инъекций во внешнюю память процессора в предопределённом режиме работы без остановки процессора. В качестве нагрузочного ПО использовалась программа «Hardware monitor», осуществляющая анализ показаний датчиков тока и температуры процессорного модуля. Показания снимались раз в секунду. Команда на

отправку телеметрической информации приходила каждые 10 секунд. Инъектор был сконфигурирован на бесконечное количество инъекций. При этом было проведено две FI компании. В первой компании мониторилась область хранения принимаемой команды на отправку телеметрической информации, для модификации поля обратного адреса SpaceWire пакета. Во второй компании мониторилась область оперативной памяти, хранящая показания датчиков тока и температуры. Длительность каждой компании составила 10 минут.

Таблица 4.7 – Результаты инъекций в предопределённом режиме без остановки процессора.

Место мониторинга	Число сбоев в SRAM	
	внесённых	исправленных
Область оперативной памяти, хранящая принятую команду на отправку телеметрической информации	60	60
Область оперативной памяти, хранящая показания датчиков тока и температуры	600	600

Все внесённые сбои оказались эффективными и были исправлены. Это связано с тем, что модифицируемая информация была актуальна для работы программного обеспечения. Высокая скорость работы инъектора позволила провести инъекции максимально прозрачно для ПО, выполняемым процессорным модулем.

В таблице 4.8 приведены результаты инъекций в режиме инъектора сбоев с остановкой процессора и предопределенным покрытием сбоев. В качестве нагрузочного ПО использовалась программа «Hardware monitor», однако инъекции вносились не в конкретную область ОЗУ, а в регистровый файл процессора, с целью промоделировать ситуацию возникновения сбоя в критичный для аппарата момент времени. Проводимые FI компании были ориентированы на проведение инъекций в момент сравнения текущих показаний датчика тока и температуры с их критическими значениями.

Соответствующий watchpoint был поставлен в регистре процессора LEON3. Инъектор ожидал момент перевода процессора в режим отладки при достижении данного watchpoint. При наступлении данного события производилась инъекция в регистровый файл. По завершению процедуры инъекции работа процессора возобновлялась.

Помимо одиночных инъекций, лист событий содержал и двойные сбои. Эффективные из них были обнаружены, однако код Хсяо не может парировать двойную ошибку, что приводило к генерации соответствующего прерывания (register_hadrware_error). В обработчике прерываний содержался код рестарта процессора при достижении данного события. Инъектор сбоев независим от процессорного модуля, и при рестарте работы процессора он продолжал свою работу по внесении сбоев в регистровый файл, ожидая наступления следующего события по watchpoint.

Таблица 4.8 – Результаты инъекций в регистровый файл

Число инъекций	Число исправленных сбоев в регистровом файле	Число обнаруженных двойных сбоев (инъектировано/обнаружено)
128	43	17/3
256	117	28/11
512	193	51/19
1024	433	112/39

Для тестирования функций сбоеустойчивости контроллера внешней памяти при случайном покрытии сбоями была использована нагрузочная программа CRC_Calc. При проведении FI компании данная программа была размещена во flash памяти. Из неё же и происходило выполнение программы (без загрузки во внешнюю SRAM память). Это было сделано для того, чтобы получить доступ ко всей области оперативной памяти. В момент выполнения вся область ОЗУ инициализировалась значением 0x10101010. Далее

выполнялось подсчёт контрольной суммы CRC инициализированного массива. При совпадении результата с ожидаемым значением подсчёт CRC в цикле повторялся вновь. При различном результате вычисления память вновь инициализировалась. При этом ПО делало вывод о наличии двойной ошибки в ОЗУ. Было проведено несколько FI компаний с разным количеством одинарных и двойных сбоев. Часть из них осуществлялась в режиме «с остановкой процессора», часть «без остановки процессора».

В таблице 4.9 приведены результаты тестирования функций сбоеустойчивости контроллера внешней памяти со случайным покрытием сбоями.

Таблица 4.9 – Результаты инъекций во внешнюю SRAM память со случайным покрытием

Тип FI компании	Число исправленных сбоев в ОЗУ (инъектировано/ обнаружено)	Число обнаруженных двойных сбоев (инъектировано/ обнаружено)
с остановкой процессора, 128 сбоев, из них 17 двойных	111/91	17/11
с остановкой процессора, 256 сбоев, из них 28 двойных	228/163	28/23
без остановки процессора, 512 сбоев, из них 51 двойных	461/359	51/39
без остановки процессора, 1024 сбоев, из них 112 двойных	912/728	112/81

При использовании предварительно загруженного листа сбоев в память инъектора количество эффективных сбоев удалось увеличить за счёт оптимального выбора момента осуществления инъекций. Результаты представлены в таблице 4.10.

Таблица 4.10 – Результаты инъекций в SRAM память из предопределённого листа сбоев

Тип FI компании	Число исправленных сбоев в ОЗУ (инъектировано/ обнаружено)	Число обнаруженных двойных сбоев (инъектировано/ обнаружено)
с остановкой процессора, 128 сбоев, из них 17 двойных	111/101	17/11
с остановкой процессора, 256 сбоев, из них 28 двойных	228/213	28/23
без остановки процессора, 512 сбоев, из них 51 двойных	461/408	51/39
без остановки процессора, 1024 сбоев, из них 112 двойных	912/892	112/81

При инъекции сбоев в ОЗУ без остановки процессора время внесения каждой инъекции несколько сократилось (в среднем на 14 тактов), ввиду отсутствия необходимости останавливать/запускать процессор на каждой инъекции. Таким образом, время инъектирования составляет не более 47 системных тактов. На период внесения инъекции доступ других устройств к ОЗУ закрыт, ввиду того, что на момент внесения инъекции механизм EDAC контроллера памяти необходимо отключать.

В таблице 4.11 приведены результаты тестирования на инъекции двойных сбоев в кэш данных (область данных) процессора с остановкой процессора со случайным покрытием сбоями. Представлены результаты четырех FI-компаний с разным числом проведенных инъекций.

Все инъекции проводимые инъектором в данных FI компаниях были ориентированы на двойные ошибки в информационном слове. При обнаружении эффективной ошибки, EDAC механизм имитировал ситуацию отсутствия попадания в кэш данных. Таким образом, искажённые данные исправлялись соответствующей им копией, хранящейся в оперативной

памяти. В качестве нагрузочного ПО использовалась программа «Single Precision C/C++ Whetstone Benchmark».

Таблица 4.11 – Результаты инъекций двойных ошибок в кэш данных (область данных) с остановкой процессора

Число инъекций	Число обнаруженных двойных сбоев
128	45
256	99
512	194
1024	441

Таким образом, в результате проведённых экспериментов были продемонстрированы следующие возможности инъектора сбоев:

– работа инъектора в режиме с/без остановки процессора;

– работа инъектора в режиме автоматической генерации параметров сбоев;

– работа в режиме чтения заданных параметров из предопределённого листа сбоев;

– внесение сбоев во внешнее ОЗУ;

– внесение сбоев в регистровый файл процессора;

– внесение сбоев в кэш память процессора;

– работа инъектора в режиме «snooping», (мониторинг транзакций записи во внешнюю память по внутрикристальной шине, либо мониторинг перехода в состояние отладки процессорного ядра по достижению установленной точки watchpoint).

Использование данных возможностей позволяет оценить функционал сбоеустойчивости процессора к сбоям в памяти. Во всех проведённых экспериментах все одиночные сбои были парированы, то есть частота катастрофических отказов оставалась равной нулю. Это подтвердило нечувствительность тестируемого процессора к одиночным событиям в

памяти. Двойные ошибки во внешнем ОЗУ и регистровом файле приводили к рестарту системы. Двойные ошибки в кэш памяти парировались (заменялись) данными из внешней памяти.

Основные результаты, приведенные в главе 4, получены в процессе создания, отладки и испытаний сбоеустойчивого процессорного модуля (ПМ) на базе процессора LEON3 для МКА «ТаблетСат-Аврора». Во время эксплуатации от разработчика и оператора МКА (компании Спутникс) замечаний к работе ПМ модуля не было, что отмечено в акте о внедрении результатов диссертационной работы, представленного в Приложении В.

4.4 Выводы по главе

Результаты проведенных экспериментальных исследований позволяют сделать следующие выводы:

1. Разработанный метод позволяет проводить инъекции с малой временной задержкой: для процессора LEON3 в режиме с остановкой процессора она составляет не более 61 системного такта, а в режиме без остановки процессора – не более 47 системных тактов.

2. Разработанный метод не требует больших ресурсов целевой системы для своей реализации: инъектор сбоев для процессора LEON3 занимает менее 2% логических блоков использованных FPGA.

3. Разработанный метод позволяет проводить длительные FI-компании, когда количество вносимых сбоев превышает 2000, что позволяет проводить статистически значимые эксперименты.

4. Разработанный метод позволяет осуществлять инъекции сбоев со случайным и предопределенным покрытием сбоями: во внутреннюю память с остановкой процессора; во внешнюю память в режимах с остановкой и без остановки процессора.

ЗАКЛЮЧЕНИЕ

В работе представлено развитие методов инъектирования сбоев в память микропроцессоров типа система на кристалле с целью тестирования их механизмов сбоеустойчивости. Предложены метод, система и методика инъектирования сбоев. Данные решения были реализованы в программируемой логической интегральной схемы для процессорного ядра LEON3, проведены экспериментальные исследования их эффективности. Основные результаты работы представлены ниже.

1. Предложен метод инъектирования сбоев в микропроцессоры типа система на кристалле с помощью встроенного аппаратного блока инъекции сбоев с использованием возможностей внутрикристального отладчика. Метод обеспечивает проведение без непосредственного участия внешнего аппаратно-программного окружения автономных экспериментов с минимальными задержками занесения сбоев в память микропроцессора. Внешний управляющий компьютер выполняет роль постобработчика проведенных экспериментов.

2. Разработана программно-аппаратная система инъекции сбоев на основе блока инъекций сбоев, встраиваемого в систему на кристалле как аппаратный сложно-функциональный блок. Она обладает низкой степенью вмешательства в аппаратные составляющие микропроцессора. Блок инъекции сбоев не требует больших ресурсов для своей реализации и позволяет проводить инъектирование как во внутреннюю, так и во внешнюю память. Он может быть использован не только для тестирования микропроцессоров в процессе их лабораторных испытаний, но и для диагностирования механизмов их сбоеустойчивости в процессе эксплуатации в составе бортовых систем космических аппаратов.

3. Разработаны методики проведения экспериментов по инъекции сбоев с помощью аппаратного инъектора, позволяющие проводить автономные

эксперименты, в которых сбои вырабатываются внутри тестируемого микропроцессора без участия внешнего программно-аппаратного окружения. Методики обеспечивают проведение инъектирования со случайным и предопределенным покрытием в режимах с остановкой процессора для тестирования сбоеустойчивости внутренней памяти, а также без остановки процессора для тестирования сбоеустойчивости внешней памяти.

4. Предложенные решения были реализованы в ПЛИС для процессора LEON3. Разработан сложно-функциональный блок инъекции сбоев, создана новая конфигурация процессора для ПЛИС, включающая разработанный блок, разработано тестовое программное обеспечение. Проведены экспериментальные исследования, которые подтвердили: реализуемость метода, широкое разнообразие видов экспериментов по инъекции сбоев; малые задержки инъектирования, определяемые лишь временными циклами работы процессора; низкие затраты ресурсов ПЛИС типа FPGA, требуемые для реализации предложенных решений.

5. Разработанный метод позволил разработать и испытать функционал сбоеустойчивости микропроцессора для бортового компьютера малого космического аппарата «ТаблетСат-Аврора».

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Осипенко, П. Одиночные сбои – вызов современных микропроцессоров/ П. Осипенко // Электронные компоненты. – 2009. –№7. – С.12-15

2 Максименко, С.Л. Анализ проблемы построения радиационно-стойких информационно-управляющих систем. / С.Л. Максименко, В.Ф. Мелехин, А.С. Филипов // Информационно-управляющие системы. – 2012. – №2. – 8с.

3 Полесский, С. Обеспечение радиационной стойкости космических аппаратов при проектировании / С.Полесский, В.Жаднов, М.Артюхова, В.Прохоров // Компоненты и технологии. – 2010. – №9. – С.93-98.

4 Чумаков, А.И. Прогнозирование локальных радиационных эффектов в ИС при воздействии факторов космического пространства / А. И.Чумаков // Микроэлектроника. – 2010. – Т.39. – №2. – С.85-90.

5 Koons, H. C. The impact of the space environment on Space systems / H. C. Koons, J. E. Mazur, R. S. Selesnick, J. B. Blake, J. F. Fennell, J. L. Roeder, P. C.Anderson // 6th Spacecraft Charging Technology Conference. – 2000. – Pp. 7-11.

6 Мироненко, Л. Повышение радиационной стойкости интегральных схем. Конструктивные методы на базе промышленной технологии / Л. Мироненко, В.Юдинцев // ЭЛЕКТРОНИКА: наука, технология, бизнес. – 2012. – №8 – С. 74-87.

7 RHBD Techniques. URL: http://www.skyflash.eu/project/radhardening/.

8 Краснюк, А.А. Особенности применения методов помехоустойчивого кодирования в суб-100-нм микросхемах памяти для космических систем / А.А. Краснюк, К.А. Петров // МЭС-2012. – Москва, ИППМ РАН, 2012. – 4 с.

9 Wiseman, B. Design and testing of SEU/SEL Immune Memory and Logic Circuits in a Commercial CMOS Process / B. Wiseman // IEEE Radiation Effects Data Workshop – 1994. – Pp. 51-55.

10 Knudsen, J. E. An Area and Power Efficient Radiation Hardened by Design Flip-Flop / J. E. Knudsen, L. T. Clark // IEEE Trans. on Nucl. Sci. – 2006. – vol. 53, no. 6. – Pp. 3392-3399.

11 Balasubramanian, A. RHBD Techniques for Mitigating Effects of Single-Event Hits Using Guard-Gates / A. Balasubramanian, B.L. Bhuva, J.D. Black, L.W. Massengill // IEEE TNS. – 2005. – vol. 52, no. 6. – Pp 2531-2535.

12 Золотарев, В.В. Помехоустойчивое кодирование. Методы и алгоритмы. Справочник / В.В.Золотарев, Г.В. Овечкин. – М.: Горячая линия - Телеком, 2004, – 126с.

13 Arlat, J. Fault Injection for the Experimental Validation of Fault-Tolerant Systems / J. Arlat // Workshop Fault-Tolerant Systems, Kyoto, Japan, IEICE. – 1992. – Pp.33-40.

14 IEC 61508 Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems URL: http://www.iec.ch/functionalsafety/standards/page2.html.

15 Elks, C. R. Development of a Fault Injection-Based Dependability Assessment Methodology for Digital I&C Systems // C. R. Elks, N. J. George, M. A. Reynolds, M. Miklo, C. Berger, S. Bingham, M. Sekhar, B. W. Johnson // NUREG/CR-7151, United States Nuclear Regulatory Commission. – vol. 1. – 2012 – 201p.

16 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. – no2. – 1990. – Pp. 166 – 182.

17 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. – Pp. 11-33.

18 Benso, A. The Art of Fault Injection / A. Benso , S. Di Carl // CEAI. – vol 13. –no 4. – 2011. – pp. 9-18.

19 Hsueh, M. Fault Injection Techniques and Tools / M. -C. Hsueh, T.K. Tsai, R.K. Iyer. // Computer. – 1997. – Vol. 30 – pp. 75-82.

20 Arlat, J. Comparison of Physical and Software-Implemented Fault Injection Techniques / J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, G. H. Leber. // IEEE Transactions on Computers. – 2003. – no. 52. – Pp. 1115-1133.

21 Ziade, H. A Survey on Fault Injection Techniques / H. Ziade, R. Ayoubi, R. Velazco // The International Arab Journal of Information Technology. – 2004. – №2. – vol. 1. – Pp.171-186.

22 Чумаков, А.И. Действие космической радиации на ИМС. −М.: Радио и связь, 2004. − 320 с.

23 Чумаков, А.И. Возможности использования локального лазерного излучения для моделирования эффектов от воздействия отдельных ядерных частиц в ИМС./ А.И. Чумаков, А.Н. Егоров, О.Б. Маврицкий, А.В. Яненко // Микроэлектроника. − 2004. − Т. 33. − № 2. − С. 128 – 133.

24 Яненко, А.В. Сравнительный анализ испытаний электронной компонентной базы на стойкость к воздействию отдельных ядерных частиц на лазерных имитаторах и ускорителях ионов / Яненко А.В., Чумаков А.И, Печенкин А.А., Савченков Д. В., Тараксин А.С., Васильев А.Л. // Спецтехника и связь. − 2011. − № 4-5. – С. 4-7.

25 Vargas, F. On the Proposition of an EMI-Based Fault Injection Approach / F. Vargas, D.E. Cavalcante, E. Gatti, D. Prestes, D. Lupi. // 11th IEEE International On-Line Testing Symposium (IOLTS 2005). − Stevenson, Washington: IEEE, 2005. − Pp. 207-208.

26 High Intensity Radiated Fields (HIRF) [Электронный ресурс] // Lab. National Aeronautics and Space Administration. Electromagnetics and Sensor Branch. − 2015. − Режим доступа: http://electromagnetics.larc.nasa.gov/facilities/hirf.htm.

27 Ramachandran, P. Statistical Fault Injection. / P. Ramachandran, P. Kudva, J. Kellington, J. Schumann, P. Sanda // 38th IEEE/IFIP Internation Conference on Dependable Systems Networks − 2008. − Pp. 122-127.

28 Tummeltshammer, P. On the Role of the Power Supply as an Entry for Common Cause Faults / P. Tummeltshammer, An Steininger // 12th International Symposium on Design and Diagnostics of Electronic Circuits & Systems. Liberec, Czech Republic: IEEE − 2009. − Pp. 152-157.

29 Arlat, J. Experimental evaluation of the fault tolerance of an atomic multicast system / Arlat, J., M. Aguera, Y. Crouzet, J. -C. Fabre, E. Martins, and D. Powell // IEEE transactions on reliability − 1990. − vol. 39. − no. 4 − Pp. 455-467.

30 Madeira, H. RIFLE: A GeneralPurpose Pin-Level Fault Injector / H. Madeira, M.Rela, F.Moreira, J.Silva // Proc. First European Dependable Computing Conference − Berlin, Germany, October 1994. − pp 199-216.

31 On Chip Debug [Электронный ресурс] // ASSET InterTech. − 2015. − Режим доступа: http://www.asset-intertech.com/Technologies/On-Chip-Debug.

32 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. − Pp. 575-582.

33 Kanawati, G. Ferrari: A Tool for the Validation of System Dependability Properties / G.A. Kanawati, N.A. Kanawati, J.A. Abraham // 22nd International Symposium on Fault-Tolerant Computing − 1992. − Pp. 336-344.

34 Kao, W.-I. Define: A Distributed Fault Injection and Monitoring Environment / W.-I. Kao, R.K. Iyer // In Fault-Tolerant Parallel and Distributed

Systems, by D. Avresky, eds. D. Pradhan. IEEE Computer Society − 1995. − Pp. 252-259.

35 Tsai, T. An Approach Towards Benchmarking of Fault-Tolerant Commercial Systems / T.K. Tsai, R.K. Iyer, D. Jewitt // 26th International Symposium on Fault Tolerant Computing − 1996. − Pp. 314-323.

36 Han, S. Doctor: An Integrated Software Fault Injection Environment for Distributed Real-Time Systems / Han, K.G. Shin, H.A. Rosenberg // International Computer Performance and Dependability Symposium. Erlangen, Germany − 1995. − Pp. 204-113.

37 Carreira, J. Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers / J. Carreira, H. Madeira, J.G. Silva // IEEE Transactions on Software Engineering − 1998. − Pp. 125-136.

38 Dasilva, A. Exhaustif: A Fault Injection Tool for Distributed Heterogeneous Embedded Systems / A. Dasilva, J.-F. Martinez, L. Lopez, A.-B. Garcia, L. Redondo // Proceedings of the 2007 Euro American Conference on Telematics and Inforamtion Systems − 2007. − P. 17

39 Assaf, M. Hardware and Software Co-Design in space Compaction of Digital Circuits / M. Assaf, S. Das, E. Petriu, L. Lin, C. Jin, D. Biswas, V. Groza, M. Sahinoglu // IEEE Instrumentation Measurement Techniques Conference − 2004. – Pp. 1472-1477.

40 Das, S. An Improved Fault Simulation Approach Based on Verilog with Application to ISCAS Benchmark Circuits / S.R. Das, S. Mukherjee, E.M. Petriu, M.H. Assaf, M. Sahinoglu, W.-B. Jone // Instrumentation and Measurement Technology Conference (IMTC 2006) − 2006. − Pp. 24-27.

41 Simulate anything, chip to system [Электронный ресурс] // WIND RIVER SIMICS − 2015. − Режим доступа: http://www.windriver.com/products/simics/.

42 Bastien, B. A Technique for Performing Fault Injection Using Simics UVA-CSCS-SFI-001 // B. Bastien, B.W. Johnson // Charlottesville: University of Virginia – 2004. – 117p.

43 Kwang-Ting, C. Fault Emulation: A New Methodology for Fault Grading / C. Kwang-Ting, H. Shi-Yu, d. Wei-Jin // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems – 1999. – Pp. 1487-1495.

44 Civera, P. New Techniques for Efficiently Assessing Reliability of SOCS / P. Civera, L. Macchiarulo, M. Rebaudengo, m. Sonza Reorda, M. Violante // Microelectronics Journal – vol 34. – 2003. – Pp. 53-61.

45 Antoni, L. Using Run-Time Reconfiguration for Fault Injection Applications / L. Antoni, R. Leveugle, B. Feher // IEEE Transactions on Instrumentation and Measurement – 2003. – Pp. 1468-1473.

46 Andres, D. Run-Time Reconfiguration for Emulating Transient Faults in VLSI Systems / D.D. Andres, J.C. Ruiz, D. Gil, P. Gil // International Conference on Dependable Systems and Networks – 2006. – Pp. 291-300.

47 Andres, D. Fault Emulation for Dependability Evaluation of VLSI Systems / D. D. Andres, J.C. Ruiz, D. Gil, P. Gil. // IEEE Transactions on Very Large Scale Integration (VLSI) Systems – vol. 16. – 2008. – Pp. 422 – 431.

48 Ejlali, A. Error Propagation Analysis Using FPGA Based SEU-Fault Injection / A. Ejlali, G. Miremadi // Microelectronics Reliability – vol. 48. – 2008 – Pp. 319-328.

49 Stott, D. Nftape: A Framework for Assessing Dependability in Distributed Systems with Lightweight Fault Injectors / D.T. Stott, B. Floering, D. Burke, Z. Kalbarczyk, R.K. Iyer // Computer Performance and Dependability Symposium, 2000. IPDS 2000. Proceedings. IEEE International – 2000. – Pp. 91-100.

50 Pattabiraman, K. Simplified: Symbolic Program-Level Fault Injection and Error Detection Framework / K. Pattabiraman, N. Nakka, Z. Kalbarczyk, R.

Iyer // IEEE International Conference on Dependable Systems and Networks with FTCS and DCC. DSN 2008 – 2008. – Pp. 472-481.

51 Bingham, S. Enhanced Fault Coverage Analysis Using ABVFI / S. Bingham, J. Lach // Workshop on Dependable and Secure Nanocomputing – 2009. – 6 p.

52 Vinter, J. An Overview of GOOFI-A Generic Object-Oriented Fault Injection Framework // J. Vinter, J. Aidemark, D. Skarin, R. Barbosa, P. Folkesson, J. Karlsson // Department of Computer Science and Engineering, Chalmers University of Technology 41296 Göteborg, Sweden. – 2005. – 40 p.

53 Yuste, P. Inerte: Integrated Nexus-Based Real-Time Fault Injection Tool for Embedded Systems / P. Yuste, D. deAndres, L. Lemus, J. Serrano, P. Gil // International Conference on Dependable Systems and Networks. San Francisco, CA – 2003. – Pp. 669-669.

54 Zenha-Rela, M. Exploiting the IEEE 1149.1 Standard for Software Reliability Evaluation in Space Applications / M. Zenha-Rela, J.C. Cunha, L.E. Santos, M. Gameiro, P. Goncalves, G. Alves, A. Fidalgo, P. Fortuna, R. Maia, L. Henriques, D. Costa // European Safety and Reliability Conference – 2006. – 7p.

55 Miklo, M. Design of a High Performance FPGA Based Fault Injector for Real-Time Safety-Critical Systems / M. Miklo, C. Elks, R. Williams // 22nd IEEE International Conference on Application-specific Systems, Architectures and Processors. Santa Monica, California – 2011.

56 The Nexus 5001 Forum Standard for a Global Embedded Processor Interface version 2.0 // IEEE-ISTO 5001 – 2003. – 166p.

57 IEEE Standard Test Access Port and Bondary-Scan Architecture // IEEE Std 1149.1 // – 2001. – 212p.

58 BDM Interface for Motorola 683xx MCU Usage with GDB Debugger [Электронный ресурс] // Режим доступа: http://cmp.felk.cvut.cz/~pisa/m683xx/bdm_driver.html.

59 Rebaudengo, M. “Evaluating the Fault Tolerance Capabilities of Embedded Systems via BDM” / M. Rebaudengo, M. Sonza Reorda // 17th IEEE VLSI Test Symposium, Dana Point, USA, April − 1999, Pp. 452-457.

60 Sonza Reorda, M. Fault Injectionbased Reliability Evaluation of SoPCs / M. Sonza Reorda, L. Sterpone, M. Violante, M. Portela-García, C. Lopez-Ongil, L. Entrena // 11th IEEE European Test Symposium – 2006 − Pp. 75-82.

61 Portela-Garcia, M. Fault Injection Approach for Measuring SEU Sensitivity in Complex Processors / M. Portela-Garcia, C. Lopez-Ongil, M. Garcia-Valderas, L. Entrena. A Rapid // 13th IEEE International On-Line Testing Symposium – Heraklion, Crete, Greece July 8 to July 11, 2007 – Pp.101-106.

62 Fidalgo, A. Real-time fault injection using enhanced on-chip debug infrastructures / A. Fidalgo, M. Gerigota, G. Alves, J. Ferreira // Microprocessors and Microsystems – 2011. – № 25. – Pp. 441-452

63 Fidalgo, A. A modified debugging infrastructure to assist real time fault injection campaigns / A. Fidalgo, G. Alves, J. Ferreira // 9th IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems – 2006. – Pp. 172 - 177

64 Fidalgo A. OCD-FI: on-chip debug and fault injection / A. Fidalgo, G. Alves, J. Ferreira // International Conference on Dependable Systems and Networks – 2006. – Pp. 214-219.

65 Yue-li Hu, Design of On-Chip Debug Module Based on MCU / Yue-li Hu, Ke-xin Zhang // Proceedings of HDP’07– 4p.

66 André V. Real Time Fault Injection Using Enhanced OCD – A Performance Analysis /André V. Fidalgo, Gustavo R. Alves, José M. Ferreira // Defect and Fault Tolerance in VLSI Systems, 2006. DFT'06. 21st IEEE International Symposium – 2006. – Pp. 254-264.

67 Ханов, В.Х. Обзор технических решений для разработки бортового комплекса управления типа система на кристалле для сверхмалого космического аппарата / Ханов В.Х., Бородина Т.В., Антамошкин А.Н. //

Вестник Сибирского государственного аэрокосмического университета, СибГАУ. – 2014. – № 5 (57). – С. 153-167.

68 Fiethe, B. Reconfigurable System-on-Chip Data Processing Units for Space Imaging Instruments / B. Fiethe, H. Michalik, C. Dierker, B. Osterloh, G. Zhou // Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07 – 2007 – Pp. 1-6.

69 Roselló, J. AGGA-4 – Core device for GNSS space-receivers of the next decade /J. Roselló Guasch, R. Weigand, G. Lopez Risueño, P. Silvestrin // NAVITEC – 2008 – 8p.

70 SOC Development Activities [Электронный ресурс] // ESA − 2015 − Режим доступа: http://www.esa.int/Our_Activities/Space_Engineering/Microelectronics/SOC_Development_Activities

71 P. Sinander. The COLE System-On-Chip – ESTEC Noordwijk, SAAB SPACE , 2007 – 11p

72 Шахматов А.В. Процессорный модуль типа система на кристалле для малого КА «ТаблетСат-Аврора» / Шахматов А.В., Чекмарев С.А. // Мат. научн.-техн. конференции молодых специалистов ОАО ИСС «Разработка, производство, испытания и эксплуатация космических аппаратов и систем» − 2014 – С.104-106.

73 UT699 LEON 3FT/SPARCTM V8 MicroProcessor Functional Manual // Aeroflex Colorado Springs − 2014. − 186p.

74 GRLIB IP Library User’s Manual Version 1.4.1 - b4156, // Cobham Gaisler AB − 2015. – 92p.

75 AMBA Open Specification, ARM [Электронный ресурс] // Режим доступа: http://www.arm.com/products/system-ip/amba/amba-open-specifications.php.

76 ECSS-E-ST-50-52C SpaceWire – Remote memory access protocol // ECSS Secretariat ESA-ESTEC Requirements & Standards Division − Noordwijk, The Netherlands 2010. – 109p.

77 Gaisler, J. A Portable and Fault-Tolerant Microprocessor Based on the SPARC V8 Architecture / J. Gaisler // Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference – 2002. – Pp. 409-415.

78 Gaisler, J. LEON3-FT-RTAX SEU test results / J. Gaisler // Gaisler Research – 2005 – 8p.

79 IEEE Standard 1076-1993. Standard VHDL Language Reference Manual (ANSI) – 273p.

80 ГОСТ Р 50754−95 Язык описания аппаратуры цифровых систем VHDL. Описание языка – 142с.

81 Чекмарёв, С.А. Способ и система инъекции ошибок для тестирования сбоеустойчивых процессоров бортовых систем космических аппаратов. / С.А. Чекмарёв // Вестник Сибирского государственного аэрокосмического университета, СибГАУ. – 2014. –№ 4(56). – С. 132–138.

82 Chekmarev S.A. Modification of Fault Injection Method via On-Chip Debugging for Processor Cores of Systems-On-Chip / S.A. Chekmarev, V.Kh. Khanov, O.A. Antamoshkin // 2015 International Siberian Conference on Control and Communications (SibCon), Proceedings. – Russia, Omsk, 2015.

83 Чекмарёв С.А. Инъекция сбоев в процессорные ядра систем на кристалле методом внутрикристальной отладки в реальном времени / С.А. Чекмарёв // Современные проблемы радиоэлектроники: сб. науч. тр. [Электронный ресурс]. – Красноярск: Сиб. федер. ун-т, 2015.– С. 235-239.

84 Осипенко П.Н. Исследование архитектурной чувствительности к сбоям с использованием метода статистического внесения сбоев / Осипенко П.Н., Антонов А.А., Левадский С.А // Программные продукты и системы – №4. – 2010. – С.12-15.

85 РД 134-0139-2005. Методы оценки стойкости к воздействию заряженных частиц космического пространства по одиночным сбоям и отказам.

86 Ханов, В.Х. Разработка аппаратуры системы информационного обмена бортового комплекса управления малого космического аппарата / Ханов В.Х., Шахматов А.В., Чекмарев С. А., Вергазов М.Ю., Лукин Ф. А. // Вестник Сибирского государственного аэрокосмического университета, СибГАУ. – 2013. – № 3 (49). – С. 149-153.

87 Чекмарёв, С.А. Моделирование бортового компьютера на базе открытых IP-блоков для малых и сверхмалых космических аппаратов / Чекмарёв С.А., Вергазов М.Ю., Лукин Ф.А., Шахматов А.В., Ханов В.Х. // Вестник Сибирского государственного аэрокосмического университета, СибГАУ – 2011. – № 2 (35). – С. 141-145.

88 Потапов, А.В. Микроспутник "Таблетсат-Аврора": прошёл год со дня запуска на орбиту / Потапов А.В., Карпенко С.О., Попов А.В., Ивлев Н.А., Сивков А.С., Власкин А.Л., Жумаев З.С., Андреенков Д.В. // Исследования солнечно-земных связей: Материалы научной сессии Секции солнечно-земных связей Совета по космосу Российской академии наук – М.: ИКИ РАН – 2015. С. 162–174.

89 Козлов, И.В. Бортовой вычислительный комплекс для негерметичных долгоресурсных КА / Козлов, И.В. // Вестник Сибирского государственного аэрокосмического университета, СибГАУ. – 2013. – № 6 (52). – С. 89-93.

90 LEON3 Processor [Электронный ресурс] // Aeroflex Gaisler − 2014. − Режим доступа: http://www.gaisler.com/index.php/products/processors/leon3

91 ProASIC3L FPGAs [Электронный ресурс] // Microsemi − 2015. − Режим доступа: http://www.microsemi.com/products/fpga-soc/fpga/proasic3l

92 Hsiao. M. A class of optimal minimum odd-weight column SEC-DED codes / M. Y. Hsiao // IBM J. Res. Develop. – 1970. – vol. 14 – no. 4 – Pp. 395–401.

93 Чекмарёв С.А. Ханов В.Х. Проектирование системы инъекции ошибок для отработки сбоеустойчивых процессоров бортовых систем малого

космического аппарата. // В мат. XVIII Междунар. научн. конф. «Решетневские чтения» / СибГАУ. – Красноярск , 2014. Т. 1. – С. 254-255.

94 ModelSim PE Student Edition // mentor graphics − 2015. − Режим доступа: http://www.mentor.com/company/higher_ed/modelsim-student-edition.

95 Чекмарёв, С.А. Разработка генератора одиночных сбоев в памяти процессора бортового компьютера космического аппарата / Чекмарёв С.А., Ханов В.Х. // Вестник Сибирского государственного аэрокосмического университета, СибГАУ –2012. – № 6 (46). – С. 229-231.

96 Linear Feedback Shift Registers [Электронный ресурс] // New Wave Instruments − 2015. − Режим доступа: http://www.newwaveinstruments.com/resources/articles/m_sequence_linear_feedback_shift_register_lfsr.htm

97 Чекмарев, С.А. Модификация IP-ядра «LEON2 Memory Controller» с целью добавления функционала отказоустойчивости / С.А. Чекмарев // В мат. III научно технической конференции «Разработка, производство, испытания и эксплуатация космических аппаратов и систем» ОАО «ИСС». –2014.– С. 102-103

98 Andreas, M. Principles of Functional Verification / M. Andreas – Newnes, 2003 – 217p.

99 Shakhmatov A.V. A functional verification system of IP-blocks in network protocols / Shakhmatov A.V., Khanov V.Kh., Chekmarev S.A. // 12TH International Conference On Actual Problems Of Electronic Instrument Engineering Proceedings (APEIE-2014), Novosibirsk, NSTU – 2014. – vol.1. – Pp. 247-251.

100 Bare-C Cross-Compiler System for LEON3/4 gcc-3.4.4 and gcc 4.4.2 [Электронный ресурс] // Aeroflex Gaisler − 2014. − Режим доступа: http://www.gaisler.com/anonftp/bcc/src/.

101 RTEMS – REAL TIME OPERATING SYSTEM [Электронный ресурс] − 2015. − Режим доступа: https://www.rtems.org/.

102 Ханов, В.Х. Программа для тестирования процессоров с помощью инъекции одиночных сбоев во внутреннюю память / Ханов В.Х., Чекмарёв С.А. // Свидетельство о государственной регистрации программы для ЭВМ №2015614979.; зарег. в Реестре программ для ЭВМ 05.05.2015.

Приложение А

Конфигурационные настройки целевой системы

(обязательное)

Данное приложение содержит конфигурационные настройки СнК-процессора LEON3, необходимые для включения инъектора сбоев.

В таблице А.1 представлены настройки IP-ядра процессора LEON3.

Таблица А.1 – Конфигурационные настройки IP-ядра LEON3

Наименование	Функция	Значение
hindex	индекс AHB мастера	0
fabtech	технология сборки	apa3 (proasic3e)
memtech	библиотека памяти для регистрового файла и кэша	apa3 (proasic3e)
nwindows	количество регистровых окон	8
dsu	синтезировать DSU интерфейс	1 (да)
fpu	тип FPU	0 (нет FPU)
v8	синтезировать аппаратный умножитель/делитель	1 (да)
cp	синтезировать сопроцессор	0 (нет)
mac	Синтезировать поддержку SPARC v8e инструкций (SMAC/UMAC)	1 (да)
nwp	количество контрольных точек	0
icen	синтезировать кэш инструкций	1
irepl	политика замены данных в кэш инструкций	0 (LRU - вытеснение давно неиспользуемых данных)
isets	количество комплектов памяти для кэша инструкций	1
ilinesize	размер строки кэш инструкций	8 слов в одной строке
isetsize	размер комплекта памяти для кэш инструкций	4 кбайт
isetlock	разрешить блокирование строки кэш инструкций	0 (нет)
dcen	синтезировать кэш данных	1 (да)
drepl	политика замены данных в кэш памяти	0 (LRU - вытеснение давно неиспользуемых данных)
dsets	количество комплектов памяти для кэша данных	1
dlinesize	размер строки кэша данных	4 слова в строке
dsetsize	размер комплекта памяти для кэша данных	4 кбайт
dsetlock	разрешить блокирование строки кэша данных	0 (нет)
dsnoop	включить контроль за изменением данных в памяти	1 (включить)

Наименование	Функция	Значение
mmuen	синтезировать блок управления памятью (MMU)	1 (синтезировать)
itlbnum	количество записей инструкций в буфере ассациотивной трансляции (TLB)	8 записей
dtlbnum	количество записей данных в буфере ассациотивной трансляции (TLB)	8 записей
tlb_type	тип TLB	общая TLB с медленной записью
tlb_rep	политика замены данных в TLB	1 (случайная замена)
disas	выводить на консоль дизассемблированные инструкции в VHDL симуляторе	0 (нет)
tbuf	размер буфера трассировки инструкций	2 кбайт
pwd	синтезировать режим включения пониженного энергопотребления	1 (поддерживается)
rstaddr	стартовый адрес после выполнения операции сброса	0x0 (адрес FLASH памяти)
smp	синтезировать поддержку многопроцессорного функционала	0 (нет)
mmupgsz	размер страницы MMU	0 (4 кбайт)
bp	включить предсказание ветвлений	1 (включить)

Как следует из таблицы А.1 IP ядро LEON3 настроено для синтеза в ПЛИС A3PE3000L, имеет раздельные кэш инструкций и кэш данных, поддерживает аппаратный блок управления памятью, аппаратные блоки умножения/деления, имеет возможность включения режима пониженного энергопотребления и не имеет аппаратного FPU. Код включения IP-ядра процессора LEON3 в состав VHDL-модели ПМ представлен в листинге А.1.

```
u0 : leon3s		-- LEON3 processor
generic map (0, fabtech, memtech, CFG_NWIN, CFG_DSU, CFG_FPU, CFG_V8,
	0, CFG_MAC, pclow, 0, CFG_NWP, CFG_ICEN, CFG_IREPL, CFG_ISETS, CFG_ILINE, CFG_ISETSZ,
	CFG_ILOCK, CFG_DCEN, CFG_DREPL, CFG_DSETS, CFG_DLINE, CFG_DSETSZ, CFG_DLOCK,
	CFG_DSNOOP, CFG_ILRAMEN, CFG_ILRAMSZ, CFG_ILRAMADDR, CFG_DLRAMEN,
	CFG_DLRAMSZ, CFG_DLRAMADDR, CFG_MMUEN, CFG_ITLBNUM, CFG_DTLBNUM,
	CFG_TLB_TYPE, CFG_TLB_REP, CFG_LDDEL, disas, CFG_ITBSZ, CFG_PWD, CFG_SVT,
	CFG_RSTADDR, CFG_NCPU-1, CFG_DFIXED, CFG_SCAN, CFG_MMU_PAGE)
	  port map (clkm, rstn_wdog, ahbmi, ahbmo(0), ahbsi, ahbso, irqi(0), irqo(0), dbgi(0), dbgo(0));
```

Листинг А.1 – Код включения IP-ядра LEON3 в состав VHDL-модели ПМ

Сигнал rstn_wdog определяет общий сброс системы при его установки в значение '0". Последовательность clkm – импульсы, генерируемые тактовым генератором с частотой 25 Мгц. Сигналы ahbmi, ahbmo(0), ahbsi, ahbso – сигналы, соответствующие входным/выходным сигналам шины AMBA AHB (master/slave). Сигналы irqi(0), irqo(0) – сигналы, взаимодействия с контроллером прерываний. Индекс '0' соответствует индексу процессора LEON3. Сигналы dbgi(0), dbgo(0) – сигналы, организующие взаимодействие по DSU интерфейсу.

IP-ядро внутрисхемного отладчика процессора DSU содержит настройки, представленные в таблице А.2.

Таблица А.2– Конфигурационные настройки IP-ядра DSU

Наименование	Функция	Значение
hindex	AHB индекс на шине AMBA	2
haddr	AHB slave адрес на шине AMBA	0x900
ncpu	количество поддерживаемых процессоров	1
tech	технология памяти, выбираемая при синтезе буфера	apa3
kbytes	размер буферной памяти	0 (отключён)

IP ядро DSU является ведомым устройством на шине AMBA AHB с индексом 2. Адрес начального регистра соответствует значению 0x80000900. В приведённой конфигурации DSU взаимодействует только с одним процессором LEON3. Использование буферной памяти отключено. Код включения IP-ядра DSU в состав VHDL-модели представлен в листинге А.2.

```
dsugen : if CFG_DSU = 1 generate
    dsu0 : dsu3                         -- LEON3 Debug Support Unit
    generic map (hindex => 2, haddr => 16#900#, hmask => 16#F00#,
      ncpu => CFG_NCPU, tbits => 30, tech => memtech, irq => 0, kbytes => CFG_ATBSZ)
    port map (rstn_wdog, clkm, ahbmi, ahbsi, ahbso(2), dbgo, dbgi, dsui, dsuo);
        dsui.enable <= '1';
  end generate;
end generate;
```

Листинг А.2 – Код включения IP-ядра DSU в состав VHDL-модели.

IP-ядро инъектора сбоев INJ_SEU содержит настройки представленные в таблице А.3.

Таблица А.3 – Конфигурационные настройки IP-ядра INJ_SEU

Наименование	Функция	Значение
abits_list_seu	определяет размер листа сбоев	7
abits_rf	характеризует размер регистрового файла процессора	8
tech	технология памяти, выбираемая при синтезе буфера	0
hindex	индекс на шине AMBA AHB (master)	3
pindex	индекс на шине AMBA APB	12
slvndx	индекс на шине AMBA AHB (slave)	1
haddr	начальный адрес на шине AMBA AHB	0xd00
hmask	определяет адрес на шине AMBA AHB	0xf00
paddr	начальный адрес на шине AMBA APB	12
pmask	определяет адрес на шине AMBA APB	0xfff
pirq	определяет номер генерируемого блоком прерывания	12
dsu_start_addr	характеризует начальный адрес DSU интерфейса	0x900

IP-ядро INJ_SEU является ведущим устройством с индексом 3. Его адрес на шине AMBA AHB соответствует значению 0xd0000000 (по этому адресу осуществляется доступ к листу сбоев и накопителю результатов). Размер листа сбоев соответствует 128 строкам по 32 бита каждая. Код включения IP-ядра INJ_SEU в состав VHDL-модели представлен в листинге А.3.

```
        inj_seu0 : inj_seu
  generic map (
     abits_list_seu=>CFG_ABITS_LIST_SEU,
    dbits_list_seu=>CFG_DBITS_LIST_SEU,
    abits_rf=>CFG_ABITSRF,
    delay_width=>CFG_DELAY_WIDTH,
    tech=> fabtech,
    hindex=>3,
    pindex=>12,
    paddr=> 12,
    pmask=> 16#fff#,
```

```
    pirq => 12
)
port map(
    clk=>clkm,
    rst=> rstn,
    apbi=>apbi,
    apbo=>apbo(12),
    ahbmi=>ahbmi,
    ahbmo=>ahbmo(3),
    ahbsi=>ahbsi,
    ahbso=>ahbso(2)
  );
```

Листинг А.3 – Код включения IP-ядра INJ_SEU в состав VHDL-модели

Настройки IP-ядра контроллера шины AHB представлены в таблице А.4.

Таблица А.4 – Конфигурационные настройки IP-ядра AHB-контроллера

Наименование	Функция	Значение
rrobin	выбор алгоритма арбитража контроллера	0 (фиксированный приоритет)
split	разрешить поддержку расщеплённых транзакций	0 (не поддерживается)
defmast	AHB-master по умолчанию	0
nahbm	количество AHB-мастеров	4
nahbs	количество AHB-slaves	4
fixbrst	включить поддержку фиксированной длины очередей	0 (не поддерживается)
debug	выводить на консоль конфигурацию	2 (все IP-ядра)
fpnpen	обеспечивает полное декодирование PNP записей	0
icheck	проверить индексы устройств на шине AMBA AHB	1 (проверить)
enbusmon	включить монитор шины AMBA	0 (отключить)
mcheck	проверить, есть ли проблемы на стыке областей памяти различных устройств (для моделирования)	1 (осуществлять проверку)
ccheck	выполнять логические проверки по записям PNP информации (для моделирования).	1 (осуществлять проверку)

IP ядро AHB-контроллера в данной конфигурации настроено для работы 4-х ведущих, 4-х ведомых устройств. Контроллер работает по алгоритму фиксированного арбитража: наивысший приоритет имеет устройство с наименьшим индексом. По умолчанию наивысший приоритет имеет процессор (его индекс равен нулю). Если шина в данный момент свободна, тогда шину также занимает процессор. Контроллер не поддерживает расщепленных транзакций. В режиме моделирования осуществляется проверка на корректность подключения устройств к шине AMBA AHB. Код включения IP-ядра AHB-контроллера в состав VHDL-модели представлен в листинге А.4.

```
ahb0 : ahbctrl                -- AHB arbiter/multiplexer
generic map (defmast => CFG_DEFMST, split => CFG_SPLIT,
        rrobin => CFG_RROBIN, ioaddr => CFG_AHBIO,
        ioen => IOAEN, nahbm => 4, nahbs => 4)
port map (rstn_wdog, clkm, ahbmi, ahbmo, ahbsi, ahbso);
```

Листинг А.4 – Код включения IP-ядра AHB-контроллера в состав VHDL-модели

В таблице А.5 представлены настройки IP-ядра контроллера интерфейса UART на шине AHB AHBUART.

Таблица А.5 – Конфигурационные настройки IP-ядра AHBUART

Наименование	Функция	Значение
hindex	AHB-мастер индекс	1
pindex	APB-slave индекс	2
paddr	начальный адрес на шине APB	2
pmask	поле маски на шине APB	0xfff

IP-ядро AHBUART является мастером на шине AMBA AHB. В данной реализации за ним закреплён индекс 1. Управляющие регистры данного IP-ядра расположены на шине AMBA APB. Начальный регистр будет расположен по адресу 0x80000200. Код включения IP-ядра AHBUART в состав VHDL-модели представлен в листинге А.5.

```
dcomgen : if CFG_AHB_UART = 1 generate
  dcom0: ahbuart                    -- Debug UART
  generic map (hindex => 1, pindex => 2, paddr => 2)
  port map (rstn_wdog, clkm, dui, duo, apbi, apbo(2), ahbmi, ahbmo(CFG_NCPU));
  dsurx_pad : inpad generic map (tech => padtech) port map (dsurx, dui.rxd);
  dsutx_pad : outpad generic map (tech => padtech) port map (dsutx, duo.txd);
end generate;
```

Листинг А.5 – Код включения IP-ядра AHBUART в состав VHDL-модели

Настройки IP-ядра контроллера памяти MEMCTRL представлены в таблице А.6.

Таблица А.6 – Конфигурационные настройки IP-ядра MEMCTRL

Наименование	Функция	Значение
hindex	AHB- slave индекс	0
pindex	APB-slave индекс	0
romaddr	начальный адрес PROM памяти	0x00000000
rommask	маска, определяющая адресное пространство PROM памяти	0xE00
ramaddr	начальный адрес RAM памяти	0x40000000
rammask	маска, определяющая адресное пространство RAM памяти	0xE00
paddr	поле, определяющее начальный адрес регистров MEMCTRL	0x00000000
pmask	маска, определяющая адресное на шине AMBA APB	0xfff
wprot	защита от записи памяти RAM	0
romasel	определяет размер PROM памяти	26 (8 Мбайт)
srbanks	определяет количество банков SRAM памяти	1
ram8	разрешает 8-битный режим для PROM и SRAM памяти	1
ram16	разрешает 16-битный режим для PROM и SRAM памяти	0
sden	включить поддержку SDRAM контроллера	0

IP-ядро MEMCTRL является ведомым устройством на шине AMBA AHB. В данной реализации за ним закреплён индекс 0. Управляющие регистры данного IP-ядра расположены на шине AMBA APB. Начальный регистр будет расположен по адресу 0x80000000. Включена поддержка 8-битного режима для работы с 8-битной FLASH памятью размером 8 Мбайт. Начальные адреса для FLASH и SRAM равны 0x00000000 и 0x40000000

соответственно. Код включения IP-ядра MEMCTRL в состав VHDL-модели представлен в листинге А.6.

```
mctrl0 : if CFG_MCTRL_LEON2 = 1 generate         -- LEON2 memory controller
   sr1 : mctrl generic map (hindex => 0, pindex => 0, paddr => 0,
            srbanks => 1, sden => CFG_MCTRL_SDEN, ram8 => CFG_MCTRL_RAM8BIT,
            ram16 => CFG_MCTRL_RAM16BIT, invclk => CFG_MCTRL_INVCLK,
            sepbus => CFG_MCTRL_SEPBUS, oepol => OEPOL,
            sdbits => 32 + 32*CFG_MCTRL_SD64, pageburst => CFG_MCTRL_PAGE,
   romaddr=>16#000#, rommask=>16#E00#, ioaddr=>16#200#, ramaddr=>16#400#, rammask=>16#E00#)
   port map (rstn_wdog, clkm, memi, memo, ahbsi, ahbso(0), apbi, apbo(0), wpo, sdo);
   addr_pad : outpadv generic map (width => 23, tech => padtech)
            port map (mem_address, memo.address(22 downto 0));
   rams_pad : outpadv generic map (width => 5, tech => padtech)
            port map (ramsn, memo.ramsn(4 downto 0));
   roms_pad : outpadv generic map (width => 2, tech => padtech)
            port map (romsn, memo.romsn(1 downto 0));
   oen_pad  : outpad generic map (tech => padtech)
            port map (oen, memo.oen);
   rwen_pad : outpadv generic map (width => 4, tech => padtech)
            port map (rwen, memo.wrn);
   roen_pad : outpadv generic map (width => 5, tech => padtech)
            port map (ramoen, memo.ramoen(4 downto 0));
   wri_pad  : outpad generic map (tech => padtech)
            port map (mem_writen, memo.writen);
   read_pad : outpad generic map (tech => padtech)
            port map (read, memo.read);
   iosn_pad : outpad generic map (tech => padtech)
            port map (iosn, memo.iosn);
   data_pad : iopadvv generic map (tech => padtech, width => 32, oepol => OEPOL)
     port map (mem_data, memo.data, memo.vbdrive, memi.data);
   brdyn_pad : inpad generic map (tech => padtech) port map (brdyn, memi.brdyn);
   bexcn_pad : inpad generic map (tech => padtech) port map (bexcn, memi.bexcn);
   memi.writen <= '1'; memi.wrn <= "1111";
   end generate;
```

Листинг А.6 – Код включения IP-ядра MEMCTRL в состав VHDL-модели

Сигналы memi, memo созданы для взаимодействия контроллера памяти с внешней памятью (PROM/SRAM). Генерируемые панели addr_pad,

rams_pad, roms_pad, oen_pad, rwen_pad, roen_pad, wri_pad, read_pad, iosn_pad, data_pad, brdyn_pad, bexcn_pad обеспечивают взаимодействие внутренних сигналов СнК со внешними сигналами (ведущими на пины ПЛИС). Сигналы apb_i, apb_o, определяют взаимодействие контроллера памяти по шине AMBA APB с другими устройствами на шине.

IP-ядро APBctrl содержит настройки представленные в таблице А.7.

Таблица А.7 – Конфигурационные настройки IP-ядра APBctrl

Наименование	Функция	Значение
hindex	индекс на шине AMBA AHB	1
nslaves	количество устройств на шине APB	15
debug	вывод отладочной информации на консоль	2
enbusmon	синтезировать AMBA APB монитор	0
mcheck	проверить, есть ли проблемы на стыке областей памяти различных устройств (для моделирования)	1
mcheck	выполнять логические проверки по записям PNP информации (для моделирования).	1

IP-ядро APBctrl является ведомым устройством с индексом 3. Его адрес на шине AMBA соответствует значению 0x80000000. Адрес остальных устройств на шине APB будет определяться следующим образом: 0x80000000 + APB_ADDR устройства на шине. Код включения IP-ядра APBctrl в состав VHDL-модели представлен в листинге А.7.

```
apb0 : apbctrl                          -- AHB/APB bridge
generic map (hindex => 1, haddr => CFG_APBADDR)
port map (rstn_wdog, clkm, ahbsi, ahbso(1), apbi, apbo );
```

Листинг А.7– Код включения IP-ядра APBctrl в состав VHDL-модели

При реализации testbench файла необходимо написать тактовые генераторы для тактирующих частот: 25 и 100 МГц.

```
constant ct : integer := 20;
constant spw : integer := 5;
```

```
clk <= not clk after ct * 1 ns;
spw <= not clk after spwct * 1 ns;
```

Кроме того в состав тестового стенда включены модели внешней SRAM и PROM памяти. Код представлен в листинге А.8.

```
 -- 8 bit prom
  prom0 : sram generic map (index => 6, abits => romdepth, fname => promfile)
          port map (address(romdepth-1 downto 0), data(31 downto 24), mem_romsn, rwen, oen);
  bank0 : for i in 0 to 1 generate
    sram0 : sram16 generic map (index => i*2, abits => 21, fname => sramfile)
port map (address(22 downto 2), data(31-i*16 downto 16-i*16), mem_ramsn(0),mem_ramsn(0),mem_ramsn(0),
rwen, mem_ramoen(0));
   end generate;
```

Листинг А.8 – Код включения внешних блоков памяти в состав VHDL-модели

Приложение Б

Свидетельство о регистрации программы

(обязательное)

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2015614979

Программа для тестирования процессоров с помощью инъекции одиночных сбоев во внутреннюю память

Правообладатель: ***Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Сибирский государственный аэрокосмический университет имени академика М.Ф. Решетнева» (СибГАУ) (RU)***

Авторы: ***Ханов Владислав Ханифович (RU),***
Чекмарёв Сергей Анатольевич (RU)

Заявка № **2015610082**
Дата поступления **12 января 2015 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***05 мая 2015 г.***

Врио руководителя Федеральной службы
по интеллектуальной собственности

Л.Л. Кирий

Приложение В

Акты внедрения

(обязательное)

СПУТНИКС

ООО «СПУТНИКС»
Тел./факс: 8 (499) 968-49-00 доб. 619
121059, Россия, г. Москва, Бережковская набережная,
дом 20, стр 6.

Исх. № 15042015-1
От 15 апреля 2015 г.

УТВЕРЖДАЮ
Генеральный директор ООО «Спутникс»
_____ А. В. Потапов
"____" _____ 2015

м.п.

Акт
о внедрении диссертационной работы Чекмарёва Сергея Анатольевича,
инженера Сибирского государственного аэрокосмического университета
имени академика М.Ф. Решетнева

Комиссия в составе:
Председатель: Попов А.В., технический директор
Члены комиссии: Сивков А. С., ведущий инженер-электронщик, Власкин А. Л., ведущий инженер-программист
составили настоящий Акт о том, что следующие результаты научно-исследовательской квалификационной работы, представленной на соискание ученой степени кандидата технических наук, использованы при создании малого космического аппарата (МКА) «ТаблетСат-Аврора»:
1. Сбоеустойчивая реализация процессорного ядра процессора Leon3.
2. Система инъекции одиночных сбоев во внутрикристальную память процессорного ядра Leon3.
3. Результаты испытаний сбоеустойчивости процессорного ядра процессора Leon3 к событиям SEU.
4. VHDL-модель системы на кристалле для сбоеустойчивого процессорного модуля МКА «ТаблетСат-Аврора».
5. Файл конфигурации FPGA для сбоеустойчивого процессорного модуля МКА «ТаблетСат-Аврора»

Данные технические решения позволили создать МКА «ТаблетСат-Аврора» с требуемым уровнем надежности и запустить его в космос. В процессе эксплуатации МКА «ТаблетСат-Аврора» к работе процессорного модуля замечаний нет, что позволяет судить о том, что сбои SEU парируются механизмом сбоеустойчивости процессора.

Председатель комиссии: _____ А.В. Попов
Члены комиссии: _____ А. С. Сивков
_____ А. Л. Власкин

федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Сибирский государственный аэрокосмический университет имени академика М.Ф. Решетнева» (СибГАУ)

Институт информатики и телекоммуникаций

УТВЕРЖДАЮ

Первый проректор – проректор по образовательной деятельности

_____ Ю. В. Ерыгин

_____ 2015

М.П.

АКТ

10.09.2015 № 3/59

г. Красноярск

Акт
об использовании результатов диссертационной работы
«Метод инъектирования сбоев для тестирования сбоеустойчивых микропроцессоров типа система на кристалле»
Чекмарёва Сергея Анатольевича
в учебном процессе

Комиссия в составе:
Председателя: Попова А.М., директора ИИТК
Членов комиссии: Колесникова С.В., заведующего кафедрой БИТ, Золотарева В.В., доцента кафедры БИТ

составили настоящий Акт о нижеследующем:

Результаты диссертационной работы С.А. Чекмарева используются в учебном процессе в Институте информатики и телекоммуникаций при проведении лабораторных работ по дисциплине «Схемотехника устройств цифровой обработки сигналов» для подготовки специалистов по специальности 10.05.02 «Информационная безопасность телекоммуникационных систем» и при выполнении дипломных и научно-исследовательских работ студентами кафедры безопасности информационных технологий Сибирского государственного аэрокосмического университета имени академика М.Ф. Решетнева.

Председатель комиссии _____ А.М. Попов

Члены комиссии:

_____ С. В. Колесников

_____ В. В. Золотарев