НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

«На правах рукопису»
УДК 004

«До захисту допущено»
Завідувач кафедри
_____ *Стіренко С.Г.*
(підпис) (ініціали, прізвище)
"06" 05 2019 р.

Магістерська дисертація

зі спеціальності: *123. Комп'ютерна інженерія*
(код та назва напряму підготовки або спеціальності)

Спеціалізація: *123. Комп'ютерні системи та мережі*

на тему: Спосіб захисту програмно-апаратних засобів програмованих логічних інтегральних схем

Виконав (-ла): студент (-ка) 6-го курсу, групи ІО-73мн
(шифр групи)

Джепбаров Ходжадурди
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник доц. д.т.н. Сергієнко А.М.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант проф. д.т.н Кулаков Ю.А.
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

Інститут/факультет Факультет інформатики та обчислювальної техніки
(повна назва)

Кафедра Кафедра обчислювальної техніки
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-науковою програмою

Спеціальність 123 Комп'ютерна інженерія
Спеціалізація Комп'ютерні системи та мережі
(код і назва)

(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сіренко С.Г.
(підпис) (ініціали, прізвище)
«06» 05 2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Джепбаров Ходжадурди
(прізвище, ім'я, по батькові)

1. Тема дисертації Спосіб захисту програмно-апаратних засобів програмованих логічних інтегральних схем
науковий керівник дисертації Сергієнко А.М., доц. д.т.н.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження процес проектування програмованих логічних інтегральних схем

4. Предмет дослідження (Вихідні дані – для магістерської дисертації за освітньо-професійною програмою) методи та засоби підвищення ефективності захисту реконфігурованих обчислювальних пристроїв

5. Перелік завдань, які потрібно розробити Систематизувати архітектурні рішення в царині захисту реконфігурованих обчислювальних пристроїв

Дослідити атаки на реконфігуровані пристрої

Розробити спосіб захисту РВС

6. Перелік графічного (ілюстративного) матеріалу Презентація на тему «Спосіб захисту програмно-апаратних засобів програмованих логічних інтегральних схем»

7. Орієнтовний перелік публікацій

8. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Кулаков Ю.А., проф. каф. ОТ		

9. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
	Аналіз існуючих архітектурних рішень реконфігурованих обчислювальних пристроїв		
	Дослідження надійності та живучості РВС		
	Дослідження здійсненостів обчислень на РВС		
	Розробка способу синтезу реконфігурацій		
	Оформлення дисертації згідно вимог		

Студент _____ (підпис) Джепбаров Ходжадурди (ініціали, прізвище)

Науковий керівник дисертації _____ (підпис) А.М. Сергієнко (ініціали, прізвище)

* Консультантом не може бути зазначено наукового керівника

Содержание

РЕФЕРАТ

Актуальность темы. Задача защиты програмнно-аппаратных средств программированных логических интегральных схем является базовой во многих системах безопасности средств вычислительной техники. Большинство известных методов защиты базируется на основе аппаратных методов защиты кода, что позволяет спроектировать безопасное устройство. Актуальность исследования способов проектирования средств защиты програмнно-аппаратных устройств на ПЛИС чрезвычайно высока, поскольку они постоянно используются в встраеваемых вычислительных системах.

Объектом исследования является процесс проектирования вычислительных систем для реализации задач защиты програмнно-аппаратных средств программированных логических интегральных схем с повышенным уровнем криптозашиты.

Предметом исследования являются методы и средства повышения эффективности функционирования вычислительных систем с повышенным уровнем зашиты на ПЛИС.

Цель работы: является усовершенствование способа защиты записи информации в ПЛИС за счет использования аппаратно обратимых функций.

Научная новизна заключается в следующемза счет использования аппаратно обратимых функций:

1. Предложен усовершенственный способ проектирования защищенных средств вычислительной техники.

Разработан аппаратный способ защиты с использованием аппаратно обратимых функций..

Практическая ценность заключается в том, что

усовершенствованный способ защиты может быть использован в создании криптостойких устройств вычислительной техники. В свою очередь, разработаная реализация усовершенственного способа защиты на VHDL может быть использована для построения встроеных средств вычислительной техники.

Апробация работы. Основные положения и результаты работы были представлены и обсуждались на научной конференции магистрантов и аспирантов.

Структура и объем работы. Магистерская диссертация состоит из введения, четырех глав и выводов.

Во введении представлена общая характеристика работы, произведена оценка современного состояния проблемы, обоснована актуальность направления исследований, сформулированы цели и задачи исследований, показано научную новизну полученных результатов и практическую ценность работы, приведены сведения об апробации результатов и их внедрение.

В *первом разделе* рассматривается описание предметной области исследований и обоснования магистерской диссертации.

Во *втором разделе* рассматривается реализация способа распознавания маркера.

В *третьем разделе* описана реализация способа защиты с использованием аппаратно обратимых функций.

В четвертом разделе описаны эксперименты с созданной системой.

В выводах представлены результаты проведенной работы.

Работа представлена на ….. листах, содержит ссылки на список использованных литературных источников.

Ключевые слова : защита информации, VHDL, ПЛИС.

Содержание

Введение

Создание программно – аппаратных средств на программируемых логических интегральных схемах (ПЛИС) связано с математической формализацией рассматриваемых процессов и явлений, превращаемых в общепринятые понятия моделей и методов. По мнению В.М. Глушковах [1] формализация процесса разработки сложной цифровой системы оперирует развивающейся во времени информационной моделью, которая содержит и исчисленческую часть. Последняя - исчисление, определяемое как совокупность правил вывода, - позволяет получать дедуктивным путем следствия из конечного множества аксиом.

По мере создания проекта меняется соотношение между информационной и исчисленческой моделями - повышается удельный вес последней - происходит математизация проекта в целях его формального описания на языках программирования или описания аппаратуры. Это способствует уменьшению временных и материальных затрат при решении проблемы проектирования.

Математизация - это возможность использования воздвигнутого на основе математического языка огромной и стройной проследовательности дедуктивных построений, как результата усилий многих поколений математиков. Поэтому формализация решаемой задачи в любой области науки дает исследователю возможность пользоваться определенной частью здания в виде соответствующего математического аппарата, благодаря чему удается сэкономить время на дедуктивные построения в целях получения необходимых выводов. В связи с этим задача определения той части математического описания, которое наиболее адекватно рассматривает проблему и с минимальными затратами формализует процесс ее решения, представляется весьма актуальной. Для решения задач проектирования полезно использовать не только новые методы и средства синтеза устройств, опирающиеся на современную элементную базу в виде программируемой

логики, но и современные языки описания аппаратуры, ориентированные на эффективное решение задач, связанных с созданием сложных цифровых систем.

Реконфигурируемость структуры вычислительного устройства является естественной потребностью при создании сложных систем, которые должны обладать высокой надежностью, гибкостью и адаптацией к решаемым задачам.

В настоящее время превалирует тенденция выполнения научных исследований и практических разработок в области реконфигурируемых вычислительных устройств (РВУ) с перспективной элементной базой (в частности, ПЛИС), которые призваны удовлетворить требования, возникающие при создании высокопроизводительных устройств обработки данных, цифровой обработки сигналов, поддержки телекоммуникаций, сопряжением комплексов и систем и т.д..

Исследования комплекса прикладных задач анализа, синтеза и оптимизации системного и логического проектирования реконфигурируемых устройств [1 - 3] показали, что сдерживающим фактором построения РВУ является отсутствие системы формальных математических моделей, которые учитывали бы характеристики особенности их функционирования, определяемые особенностями новой элементной базой (в частности, ПЛИС).

Таким образом, задача построения формализованной системы согласованных математических понятий, адекватно описывающих характерные особенности функционирования РВУ, является актуальной и важной для современного этапа развития теории проектирования вычислительных устройств.

Существует несколько практических причин применения моделей:

1. Формальная модель используется для создания технического задания.

2. Модель необходима при кооперации труда проектировщика и

пользователя. Вместо длинного и не всегда полного описания поведения системы в разных режимах проектировщик может предоставить формальную модель системы.

3. Возможность тестирования и верификации проекта через создание его моделей.

Вначале создается общая модель, описывающая функционирование всего устройства. Для ее верификации строится тест, полученные выходные реакции принимаются в качестве эталонных. Затем, следуя концепции нисходящей методики проектирования, эта модель разбивается на подсистемы. Полученный ранее тест может быть использован для проверки новой, более детализированной модели. Проект требует доработки, если полученные на данном шаге реакции не совпадают с эталонными. Иначе, детализация модели проекта будет продолжаться до тех пор, пока составляющие проект компоненты не будут соответствовать реальным устройствам.

Полученный тест может быть использован и на стадии производства для верифи-. кации реального устройства. Подразумевается, что тест проверяет все режимы, в которых созданная цифровая система будет эксплуатироваться.

4. Возможность формальной верификации схемы. Формальная реализация требует математического описания функций системы. Оно может быть выполнено в терминах формальной логики, включающей параметры времени. Формальная верификация также требует математического определения конструкций языка моделирования или обозначений, используемых для описания схем. Хотя формальная верификация в настоящее время еще широко не применяется, ведутся активные исследования в данном направлении. Уже существуют демонстрации методов формальной верификации при проектировании реальной схемы.

5. Возможность автоматического синтеза схем. Если возможно

формально описать функции системы, теоретически можно трансформировать это описание в схему, реализующую данные функции. Это позволяет сократить долю ручного труда в проектировании и уменьшить вероятность внесения ошибок в проект. Если использовалась автоматическая трансляция описания в схему, можно утверждать, что последняя соответствует исходной спецификации.

В настоящее время одним из интенсивно развивающихся направлений в микроэлектронике является PLD (programmable logicdevice - программируемые логические устройства). PLD представляют собой пустой чип. В отличие от обычных цифровых микросхем, логика работы ПЛИС определяется не на фабрике при его изготовлении, а задается посредством дополнительного программирования с помощью специальных средств программаторов и программного обеспечения. Эта технология позволяет разработать свою микросхему с собственной архитектурой.

В связи с широким использованием ПЛИС важное место занимает защита интеллектуальной собственности. Продукты интеллектуальной деятельности имеют стоимостные оценки, так как они могут быть включены в товарооборот на коммерческих условиях.

Угрозу представляет кража интеллектуальной собственности, последствиями которой являются экономические потери. Предотвратить ее можно защитив цифровые электронные устройства.

Распространение нелегального копирования программного продукта, высокая стоимость разработки ПО (современные сложные программы пишутся командой программистов), возможность извлечения информации из постоянных запоминающих устройств ПЛИС беспрепятственно,наличие недобросовестной конкуренции на рынке электроники определяет важность и актуальность защитв интеллектуальной собственности при проектировании средств вычислительной техники..

Данная работа посвящена разработке способа защиты програмнно –

аппаратных средств разработаных на базе ПЛИС.

1. ИССЛЕДОВАНИЕ МЕТОДОВ ЗАЩИТЫ ПРОГРАММНОГО ПРОДУКТА ДЛЯ ПЛИС

1.1 Анализ злономаренно созданных аппаратных стедств.

Злонамирено созданным программным обеспечением (malicious software) считается программы, разработанные с целью нарушения режима защищенного и надежного функционирование вычислительных систем (ВС). Создавать такое программное обеспечение достаточно просто и заманчиво. Это обусловливается распространением типовых архитектур компьютеров, большим количеством возможностей поражения вычислительных систем со стороны матобеспечение, а также в определенных случаях – значительной выгодой от такой деятельности. Поэтому до сих пор наибольшее внимание в научных исследованиях, организационных и технических мероприятиятиях относительно защиты ВС посвящено именно такому программному обеспечению.

Злонамирено создано аппаратное обеспечение (ЗСАО, malicious hardware) можно определить аналогично. Это аппаратное обеспечение ВС, разработанное и внедрено ради нарушение режима защищенного и надежного функционирования ВС. Очевидно ЗСАО и ЗСАО имеют много общего с вредным программным обеспечением. Поэтому классификация ЗСАО похожа на классификацию последнего.

ЗСАО типа черный ход (backdoor, trapdoor) позволяет осуществить доступ в ВС, который не обусловливается его техническими условиями. Оно может быть внедрено во время разработки ВС или ее модернизации.

ЗСАО типа уничтожающий переключатель (kill switch) – это вредный искусственный объект который дает возможность атакующим нарушить корректное функционирование аппаратуры или матобеспечения. Так же, как и черный ход, уничтожающий переключатель может быть внедренный во время разработки ВС или ее модернизации. Но вместо возможности несанкционного доступа, он вызывает отказ в виде несостоятельности

выполнить необходимые функции. Он может быть установлен как часть прошивки ПЛИС или в проект микросхемы с помощью утилит САПР микросхем, или во время изготовления микросхемы.

Вирус ПЛИС – это часть прошивки ПЛИС, которая может вызывать короткое замыкание выходов внутренних вентилей и как результат, – перегрев и выход из строя микросхемы [8]. Вирусы ПЛИС могут быть внедрены с помощью матобеспечения или даже путем коррекции топологии микросхемы ПЛИС (что маловероятное, но возможно). Хотя вирусы ПЛИС – не являются распространенным явлением, их возможное приложение следует всегда иметь в виду.

Троянская аппаратура – это широкий класс разнообразных ЗСАО. Как и троянские программы, эти ЗСАО внедряются посторонними лицами и не предназначенные для вывода из строя ВС. Результатом действия может быть нарушение нормального функционирования устройства, как, например, замедление действия, уменьшения точности или операций ввода вывода важной информации через скрытый канал. Они различаются по физическому принципу реализации, способу активизации и целевой функции [9]. Внедрить ЗСАО в микросхему можно во время ее изготовления или путем физического вмешательства в нее. В последнем случае есть возможность добраться до поверхности кристалла и средствами обработки сфокусированным ионным лучом изменить геометрию слоев кристалла, то есть удалить определенные проводники, прибавить другие проводники или откорректировать емкость, сопротивление, задержку, максимально допустимый ток. Функциональное ЗСАО выполняется путем добавления, исключения или короткого замыкания транзисторов или вентилей.

Параметрическое ЗСАО реализуется через модификацию существующих линий связи и других дискретных компонентов кристалла. Например, утонченя проводника или нарушение транзистора или резистора не влекут отказ, но существенно увеличивают достоверность отказа со

временем в результате электромиграции или приводят к существенному уменьшению производительности или точности. По размеру и количеству задействованных входов-выходов ЗСАО разделяют на малых и больших.

Большой блок ЗСАО значительно более легко активизировать и обнаружить, чем малый. Зато он имеет функциональность, которая обеспечивает решение сложных заданий, которые влекут большие потери.

Различают также сконцентрированные и распределены ЗСАО. Последние однородно распределены по площади кристалла и потому является значительно сложнее для выявления через анализ топологии. Их более легко изготовить, если количество доступных для модификации вентилей кристалла является ограниченным. С другой стороны, у сконцентрированного ЗСАО меньшая площадь, более короткие проводники и потому его тяжелее обнаружить за непрямими признаками – изменению энергопотребления, максимальной тактовой частоты в сравнении с эталонным кристаллом..

ЗСАО может быть активированным извне через скрытый канал или изнутри. Последние ЗСАО разделяются на всегда включенные и такие, которые включаются по условием. Всегда включенные ЗСАО является постоянно активированным и могут нарушить функционирование ВС в любой момент. Это, как правило, параметрическое ЗСАО.

ЗСАО, которые включаются по условию, являются неактивным пока не исполняется определенное условие. Это, например, детектирование конкретных параметров, таких как температура, давление, напряжение, уровень электромагнитного поля и тому подобное, или их комбинации. Это может быть фиксация заданного внутреннего логического состояния или состояния входных сигналов, или переполнения внутреннего счетчика. Такое ЗСАО почти “невидимое”, то есть не может быть обнаруженным до своей активизации.

За своей целевой функцией ЗСАО разделяются на такие, которые

изменяют функциональность микросхемы, модифицируют ее параметры или предназначают для передачи информации. Внедрение ЗСАО изменяет функциональность через добавление новых логических схем, или через исключение или закорачивает уже существующих

Модификация параметров – задержек, порога срабатывания, уровня усиления, тока питания и тому подобное – достигается изменением геометрии проводников и резисторов.

ЗСАО для передачи информации предназначенные для пересылки секретной информации из защищенной области наружу. Возможно также сочетание передачи информации с изменением функциональности или параметров. Например, определено изменение параметров схемы блока шифровки ВС дает возможность определить ключ шифрования через анализ измерений энергопотребления этой ВС.

1.2 Анализ возможности реализации ЗСАО в ПЛИС.

Теоретически ЗСАО может быть заложена в ПЛИС во время изготовления кристалла. Злоумышленник может находиться в компании, которая изготовляет маски или кристаллы, или монтирует их в корпуса. Он должен иметь возможность копировать, анализировать и изменять проект микросхемы. Но из-за того, что на этапе производства нет инфориации о том, где и каким образом будут использованы ПЛИС, какая именно прошивка будет реализована, внедрение ЗСАО, которое должно выполнить определенные функции, является слишком сложным заданием.

Потенциальными местами, в которых может быть подсоедино ЗСАО путем определенной коррекции масок, аппаратное микропроцессорное ядро, блоки оперативной памяти, блок ввода вывода данных, блок ввода конфигурационной последовательности и блок криптообробки этой последовательности [10].

Злоумышленник потенциально способен вставить ЗСАО в матрицу ПЛИС на этапе ее изготовления, имея надежду, что ЗСАО сработает как уничтожающий переключатель для проекта который будет на ней сконфигурирован. В расчете на этот случай, разработчик проекта может принять меры, чтобы предотвратить его ненадежную работу. Например, ответственные блоки проекту можно выполнить с тройным резервированием и рассредоточить их по кристаллу. Кроме того, система загрузки и сохранения прошивки имеет достаточно надежную защиту, чтобы быть нарушенной вмешательством со стороны ЗСАО. Чтобы выполнить такую атаку против конкретного пользователя ПЛИС, производитель должен выпустить значительно большие партии кристаллов, чем нужно. При этом ЗСАО сработает у многих пользователей и через процедуру рекламации в конце концов, будет обнаружено наличие ЗСАО.

Поэтому такой подход является ненадежным. Хотя кристалл ПЛИС и ВС, где он внедрен, является потенциально незащищенны от вмешательство, прошивка ПЛИС может быть разработана в условиях секретности. Поэтому проект для ПЛИС, если с ним не будет ознакомлен злоумышленник, является защищенным.

Так как функционирование ПЛИС основывается на информации в файле прошивки, к ней применяются стандартные методы информационной защиты. Атаки на аппаратуру с реконструированием ее проекта, которые используются для обычных микросхем, для ПЛИС является неприемлемыми потому что они приводят к разрушению прошивки. Чтобы внедрить злоумышленнику свое ЗСАО в ПЛИС, необходимо иметь начальный проект.

Информация о проекте закодирована в файле прошивки. Пусть злоумышленник завладел файлом прошивки. Невзирая на то, что правила кодировки прошивки известны производителю ПЛИС и заложенные в открытых средствах проектирования ПЛИС, до сих пор не существует сведентй о успешном реконструировании ни одного проекта на основе

такого файла прошивки.

Невзирая на это, в ответственных приложениях принимаются дополнительные мероприятия защиты. Самое простое мероприятие – это однократное программирование, когда полностью отсутствует доступ к прошивки при условии постоянной поддержки питания ПЛИС. В течение последнего десятилетие в ПЛИС применяется шифровка файла прошивки, что позволяет многократная ее загрузка в ПЛИС после выключения и включения питания без возможности несанкционированного копирования. Это дополнительно делает невозможным реконструирование проекта. Кроме того, в режиме шифрованного файла прошивки в ПЛИС запрещена частичная конфигурация, которая позволяет присоединять дополнительные части проекта. Также условием загрузка такого файла является объязательное выключение питания для уничтожения предыдущей конфигурации. Ключ шифра файла прошивки сохраняется в ПЛИС в оперативной памятиі с внешним питанием и автоматически стирается при его выключении.

Известны способы атаки на этот ключ приводят к выключению этого питания и разрушению шифра и конфигурации, поскольку ключ сохраняется в глубине кристалла под несколькими слоями диэлектрика и металлизации. Таким образом, не единственной возможностью ли вставить ЗСАО в ПЛИС есть присоединение его во время проектирования, то есть в условиях нарушенной секретности. Например, можно вставить в проект виртуальный модуль от постороннего производителя со скрытым у него ЗСАО встроить в программные средства автоматизированного проектирования программу-генератор, которая незаметно подгонит ЗСАО к проекту [10].

1.3 Скрыты каналы в ЗСАО в ПЛИС .

Скрытый канал – это средство передачи информации между объєктами

способом, не предусмотренным для пересылки информации и не разрешенным правилами политики защиты [3]. Скрытый канал может быть аналоговым или цифровым.

Аналоговый канал строят, используя разнообразные физические явления. Это, например, модулируются ток потребления, температура корпуса, яркость контрольного светодиоду, наконец, радиоволны, которые излучаются частями схемы.

Цифровой канал организуют путем статистической модуляции определенных разрядов данных, которые передаются через легальные каналы или благодаря модуляции штатных событий, например, сигналов прерывания, ошибки или неготовности блока таким образом, чтобы их поведение было правдоподобным.

Скрытый канал конструируют с учетом того, чтобы его было невозможно или трудно обнаружить на фоне помех обычными средствами противодействия. Более того, специально сконструированный ЗСАО, что реализует скрытый радиоканал, может иметь радиус действия до нескольких десятков метров, причем сигнал канала трудно обнаружить через его шумо подобную модуляцию [12]. Для надежной пересылки данных следует синхронизировать как прием, так и их передачу. Поскольку возможности ЗСАО ограничены, передача данных чаще всего является несинхронизированной. Из-за этого канал настроен на медленную передачу определенного типа данных, например, лишь криптографического ключа.

Скрытый канал часто конструируют двунаправленным. Обратное направление используется для синхронизации и активизации ЗСАО. Сложные ВС включают в себя блоки с разной степенью защиты. Тогда скрытый канал может соединять блок с высокой степенью защиты с блоками с меньшими степенями защиты. Достаточно эффективным скрытым каналом является ресурс, который разделяется между этими блоками. Одним из таких ресурсов есть кеш-память системы. Организация

скрытых каналов через таких разделены ресурсы широко употребимая при использовании вредного матобеспечения. Их приложение при использовании ЗСАО исследовано в [3].

1.4 Способы выявления ЗСАО

Можно выделить три основных подхода к выявлению ЗСАО в микросхемах. Это физический анализ топологии кристалла, тестирования с автоматической генерацией тестов и анализ сигналов непрямих каналов передачи информации [9].

Топологию кристалла анализируют с помощью автоматизированных средств, которые применяются при организации производства микросхем, например, сканирующего электронного микроскопf. При этом тестируемая микросхема сравнивается с эталонной. Такой анализ является дорогим и чрезвычайно медленным. Кроме того, микросхему следует осторожно изъять из корпуса и препарировать, что также является сложным заданием. При уменьшении проектных форм микросхем уменьшается достоверность выявления ЗСАО и растет цена оборудования.

Согласно второму подходу, используются стандартные средства тестирования микросхем на основе автоматизированной генерации тестовых последовательностей. При этом тестирование должно быть настроено на регистрацию параметрических изменений в схеме в случае поиска параметрического ЗСАО, или на выявление модификации логической схемы при поиске функциональных ЗСАО.

К сожалению, во втором случае тестирования является неэффективным через определенные причины. Известно, что задача выявления вредного программного обеспечения сводится к задаче остановки машины Тюринга и потому не может быть решенной в общем случае [3,12].

То же касается выявления ЗСАО. О сложности этой задачи

свидетельствует пример разработки троянского ЗСАО, предназначенного для создания скрытого канала микропроцессора. Два варианта ЗСАО имеют, соответственно, 959 и 1341 логического вентиля и составляют лишь 0.05% и 0.08% от общих аппаратных расходов ВС [4]. Без информации о логической схеме этих ЗСАО, места их подключения, способа и деталей активизации практически невозможно найти тестовую последовательность, которая обнаруживает наличие этих ЗСАО.

Анализ непрямих каналов истока информации (side channels) дает возможность обнаружить присутствие ЗСАО. Например, можно включить микросхему и измерять аналоговые сигналы отзыва, такие как ток потребление, инфракрасное и радиоизлучение. Через анализ потребляемого тока потенциально возможно обнаружить даже неактивировано ЗСАО, потому что схема наблюдения за условием активизации все время потребляет некоторый ток и выполняет определенные переключения, какие отличающиеся от тока и переключений оригинальной микросхемы.

Эффективность анализа непрямих каналов растет, если в микросхему были внедрены специальные модули, которые способствуют этому анализу. Это могут быть датчики напряжения, температуры в разных точках кристалла, модули измерения задержки (добавление проводников ЗСАО увеличивают задержку), а также схемы встроенного самого тестирования [2,3].

В отличие от сложного анализа, выявления ЗСАО в ПЛИС является тривиальным. Любая коррекция файла прошивки изменяет его контрольный код. Кроме того, существующие программные средства дают возможность сравнивать между собой проекты на разных стадиях проектирования и обнаруживать между ними отличия, включая нежелательные логические схемы [10].

1.5 Предупреждение внедрения ЗСАО

Вредное действие ЗСАО можно сделать невозможным или компенсировать благодаря специальным мероприятиям структурного проектирования ВС. Применение резервирования, которое противодействует уничтожающему переключателю, а также внедрение инфраструктурных модулей – параметрических датчиков, которые помогают обнаружить или противодействовать ЗСАО позволяют исключить Вредное действие ЗСАО .

Главными мероприятиями Предупреждение внедрения ЗСАО следующие.

1 Наиболее эффективным способом является пространственная и логическая изоляция ответственных блоков от остальной схемы ВС (способ moats and drawbridges – буквально – замковый ров и цепной мост). Такая изоляция существенно осложняет настройку скрытого канала и упрощает его выявление.

2 Для устранения скрытого канала через блок с раздельным доступом следует использовать механизм подтверждения обращения к такому блоку. Тогда схема мониторинга обращения будет отказывать в доступе другим блокам, включая ЗСАО, которые не имеют на это права [3].

3 Для минимизации истока информации через побочные каналы следует использовать логические схемы, в которых или минимизирован шум тока переключений, или этот шум маскируется дополнительным генератором шума. При таких условиях организация скрытого аналогового канала значительно осложняется [13].

5 Исследуя структуру и выполняемые алгоритмы существующей ВС, можно предусмотреть места и принцип действия потенциальных ЗСАО и на основе этого организовать соответствующие предупредительные мероприятия. Такое исследование проводится, как правило, путем

моделирования [2,14].

Выводы к 1 разделу

В последнее время злонамерено созданному аппаратному обеспечению (ЗСАО) в мире уделяется много внимания из-за того, что оно составляет серьезную угрозу безопасности государства и людей. Выявление или обезвреживания ЗСАО, которые встроены в ВС, является очень сложной задачей, которое осложняется с ростом степени интеграции ПЛИС в ВС

Вычислительные системы на базе ПЛИС имеют существенные преимущества в потому, что значительное усложненное внедрение ЗСАО в ПЛИС и существенно упрощают его выявление. Самый дешевый и самый надежный способ сделать невозможным внедрение ЗСАО в ПЛИС – это выполнение соответствующих мер безопасности во время разработки файла конфигурации проекта. Структурными и логическими мероприятиями, которые предупреждают внедрение ЗСАО, является проектирование с резервированием, пространственная и логическая изоляция ответственных блоков, управления доступом к разделенным ресурсам, минимизация шума переключений, внедрения инфраструктурных модулей противодействия ЗСАО.

Разработка, выявление, противодействие и предупреждение ЗСАО – это круг заданий, которые нуждаются в последующих многосторонних научно-технических исследованиях.

2. ИССЛЕДОВАНИЕ МЕТОДОВ ЗАЩИТЫ ЦИФРОВЫХ ПРОЕКТОВ И АППАРАТНЫХ УСТРОЙСТВ НА ПЛИС

2.1 Модель взаимодействия пользователя и цифрового устройства.

В случае реализации цифрових устройств на базе ПЛИС проблема защиты от несанкционированнх действий человека становится более масштабной. Выражается это в первую очередь в том, что HDL-кодн й BIT-образы конфигурации ПЛИС передаются й распространяются в открытом и незащищенном виде.

На рис. 2.1 представлена общая схема взаимодействия производителя и пользователя цифровой системи с указанием объектов пользования. Проектируемое цифровое устройство на стороне разработчика находится в доверительном окружении, в которое входят: система автоматизированного проектирования, включающая программные средства создания и редактирования проектних HDL-описаний, программные средства синтеза (генерирования BIT-образа), цифровую система на базе ПЛИС и память конфигурации PROM.

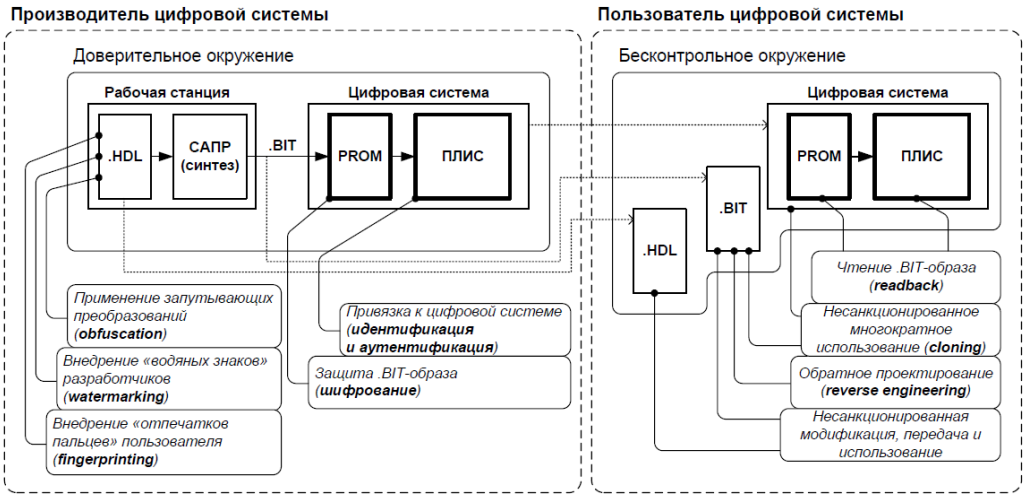


Рисунок. 2.1 Общенная схема взаимодействия производителя и пользователя цифровой системи

В доверительном окружении создаются обьектн интеллектуальной собственности, к которым относятся файлы проектных HDL-описаний, файл BIT-образа конфигурации ПЛИС, сконфигурированная цифровая система (BIT-образ записан в PROM). Перечисленнне обпекты могут передаваться пользователю (вместе либо по отдельности) в зависимости от соглашения с производителем.

Будучи переданными пользователю, обпекти интеллектуальной собственности находятся в бесконтрольном окружении, что означает наличиє возможности несанкционированнвіх действий по отношению к ним.

К несанкционированньш действиям можно отнести следующие:

— несанкционированная модификация, передача либо использование HDL-описаний (IP-компонент), повлекшие за собой нарушение авторских прав производителя цифровой системы;

— обратное проектирование BIT-образа, при котором злоумышленник осуществляет преобразование файла конфигурации ПЛИС в netlist - описание и/или иное описание, которое способствует пониманию внутренней структуры и особенностей функционирования цифровой системи;

— многократное несанкционированное использование BIT-образа , подразумевающее тиражирование цифровой системи в обход соглашений с производителем (например, изготовление более дешевой цифровой системы с идентичной ПЛИС й использование оригинального BIT-образа для ее конфигурации);

— чтение BIT-образа непосредственно из ПЛИС либо PROM сконфигурированной цифровой системы для проведення обратного проектирования.

Для предотвращения несанкционированых действий со строны пользователя применяются ряд методик.

2.2. Применение запутывающих преобразований HDL-описаний

Запутывающие преобразования, либо обфускация, — это приведение исходного текста или исполняемого кода программы к виду, сохраняющему ее функциональность, но затрудняющему анализ, понимание алгоритмов работы и модификацию при декомпиляции [5]. С точки зрения HDL-описаний под обфускацией можно понимать следующие два типа преобразований: лексическую обфускацию и функциональную обфускацию.

Предположим, что имеется исходное HDL - описание V, которое путем применения к нему всех последовательностей проектних действий DD преобразуется в цифровую схему S: DD(V) —> S. Применив обфусцирующие преобразования O к V, получаем описание V': O(V) —> V', которое в результате преобразуется в идентичную схему S: DD(V') —> S, при V'обладает меньшей наглядностью й является более сложным для чтения й понимания даже профессиональным проектировщиком.

Подобный вид преобразования является лексической обфускацией и основан на изменении форматирования исходного HDL-описания, способствующий усложнению его зрительного восприятия. Однако, по сути, все языковые конструкции при этом не подвергаются модификациям, что и влечет за собой получение идентичной схемы S.

К возможным изменениям формата исходного описания V можно отнести следующие:

— удаление всех комментариев;

—удаление/добавление символов-разделителей между лексемами HDL-описания;

—замена имен пользовательских идентификаторов на идентификаторы с произвольными/нечитабельными именами.

Так, удаление комментариев само по себе является первой преградой для быстрого понимания исходного HDL-описания, а форматирование путем удаления таких разделителей, как конец строки (EOF), делает описание

невосприимчивым к прочтению. При составлении HDL-описаний проектировщики часто используют имена идентификаторов, которые несут некую смысловую нагрузку с точки зрения их назначения. Например, сигналы синхронизации часто объявляются как CLK, c1ock и т.п., что уже является достаточной информацией для понимания их назначения. Замена исходных имен идентификаторов на менее читабельные еще больше затрудняет как восприятие, так и понимание HDL-описания.

Кроме лексической обфускации к HDL-описаниям можно применять обфускацию потока выполнения [5], которая для языков программирования означает преобразование графа передачи управления, достигаемое за счет локальных преобразований кода либо эквивалентных преобразований алгоритма. Например, для языка VHDL запись параллельного оператора присваивания может быть заменена на структурное описание сторонней компоненты либо на использование функции [6]. А свойство параллельности операторов языка VHDL, представленных в теле блока architecture, может быть использовано для изменения порядка их следования в исходном описании.

Прочтение HDL-описания, подвергнутого лексической обфускации, является весьма затруднительным. Однако, какова бы ни была степень сложности подобных преобразований, в результате HDL-описание остается синтезируемым, что открывает возможность понять структуру и функциональные особенности устройства по результатам синтеза.

В связи с этим интерес вызывает функциональная обфускация, под которой можно понимать процесс получения эквивалентной цифровой схемы. Пусть дано исходное описание V, которое регламентирует функционирование и/или структуру цифрового устройства S: DD(V) —> S. Преобразуем V таким образом, что полученное после синтеза устройство S" будет являться функциональным эквивалентом устройства S: DD(O(V")) —> S".

Предположим, что схема цифрового устройства и описывается следующим выражением:

$S = \{IP_0,\ldots, IP_{i-1}; OP_0,\ldots, OP_{i-1}; B_0,\ldots,B_{k-1}; L_0,\ldots,L_{k-1}\}$

где IP — множество входных портов, OP — множество выходных портов, B — множество функциональных блоков из которых состоит схема устройства, L — множество проводящих линий, соединяющих внутренние блоки и порты.

Определим функциональность устройства S как зависимость значений на выходных портах OP от значений на входных портах IP, внутренних блоков B и проводящих линий L:

OP= Fs(IP,.B,L).

Предположим, что существует другое цифровое устройство S', схема которого описывается как

$S' = \{IP_0,\ldots, IP_{i-1}; OP_0,\ldots, OP_{i-1}; B'_0,\ldots,B'_{k-1}; L'_0,\ldots,L'_{k-1}\}$

при этом множество блоков B' и множество линий L' может не совпадать с аналогичными множествами устройства S.

Устройства S и S' будут являться функционально эквивалентными, если выполняется следующее равенство: Fs(IP, B.L) = Fs'(IP. B'. L')

Таким образом задача обфускации исходных HDL-описаний сводится к затруднению понимания структуры и функциональных особенностей разработанной схемы. Если к рассмотренному примеру функциональной обфускации применить лексическую обфускацию, то для понимания исходного VHDL - кода и синтезируемой схемы злоумышленнику необходимо приложить усилия, эквивалентные затраченным проектировщиком при создании проектного описания.

Однако применение обфусцирующих преобразований не является препятствием для нелегального распространения исходного HDL-описания с дальнейшей реализацией на ПЛИС, тем самым нарушаются права интеллектуальной собственности проектировщика.

2.3. Анализ методики внедрение «водяных знаков» и «отпечатков пальцев» в HDL-описания.

Технология «водяных знаков» широко распространена для защиты прав интеллектуальной собственности на различные объектах цифровых технологий, в том числе и на программное обеспечение [5]. В общем случае «водяной знак» внедряется разработчиком в объект его интеллектуальной собственности для возможности в дальнейшем доказать свое право интеллектуальной собственности на данный объект.

«Отпечаток пальца» по сути также является «водяным знаком», несущим информацию не только о правообладателе, но и о пользователе, которому легально была предоставлена возможность пользования данной копией объекта интеллектуальной собственности. Рассмотрим, каким образом можно осуществлять внедрение «водяных знаков» в проектные HDL-описания.

Пусть дано исходное HDL-описание V цифрового устройства S. Пусть K есть «водяной знак», а WM есть функция внедрения K в V:

Передаваемое HDL-описание с внедренным «водяным знаком» Vk должно обладать рядом свойств, наиболее значимыми из которых являются следующие:

— K не должно присутствовать в Vk явным образом;

— синтезируемая схема из Vk также должна содержать K: DD(Vk) —> Sk;

— внедренный K не должен существенным образом влиять на функционирование, аппаратные затраты и временные характеристики цифрового устройства;

— K должен быть подобран таким образом, чтобы исключить случайное появление его копии в Vk и в Sk;

— проектировщик, осуществивший внедрение K, должен иметь возможность доказать, что именно он и никто другой осуществил внедрение

WM(V,K) — > Vk.

Первое свойство необходимо для исключения возможности нахождения и удаления «водяного знака» из текста HDL-описания. Если К все таки найден, то его удаление либо изменение должно отразиться на функционировании устройства. Попытка удаления, изменения или подмены водяного знака должна быть равносильна перепроектированию цифрового устройства. Внедренный «водяной знак» должен быть прозрачным для средств синтеза для исключения возможности его удаления путем обратного проектирования RE, например при попытке преобразовать B1T-образ в netlist-описание и далее в HDL-описание с последующим синтезом: RE(Sk)) — > Vk.

Для возможности доказательства авторства внедренного «водяного знака» проектировщик может использовать значение К в качестве результата шифрования своих данных (например копирайт-строку) с помощью секретного ключа, значение которого известно лишь ему. На рис. 5.6 представлена обобщенная схема внедрения «водяного знака» и процесса доказательства авторства.

Многие разработанные методы и алгоритмы внедрения «водяных знаков» для программного обеспечения могут быть применены и к HDL-описаниям [5].

2.4. Методы аутентификации и идентификации цифровых устройств, на ПЛИС

Многие производители интегральных схем решают задачу идентификации СБИС путем реализации специальных регистров хранения уникальных идентификаторов (серийных номеров), значения которых, как правило, единожды задаются на этапе производства и в последующем использовании доступны только для чтения.

Наличие такого рода идентификаторов позволяет проектировщикам цифровых систем эффективно решать многие задачи: адресация СБИС,

подключенных к единой информационной магистрали; использование идентификаторов в качестве открытого ключа при реализации алгоритмов шифрования; реализация методов и алгоритмов защиты от несанкционированного использования и т.д.

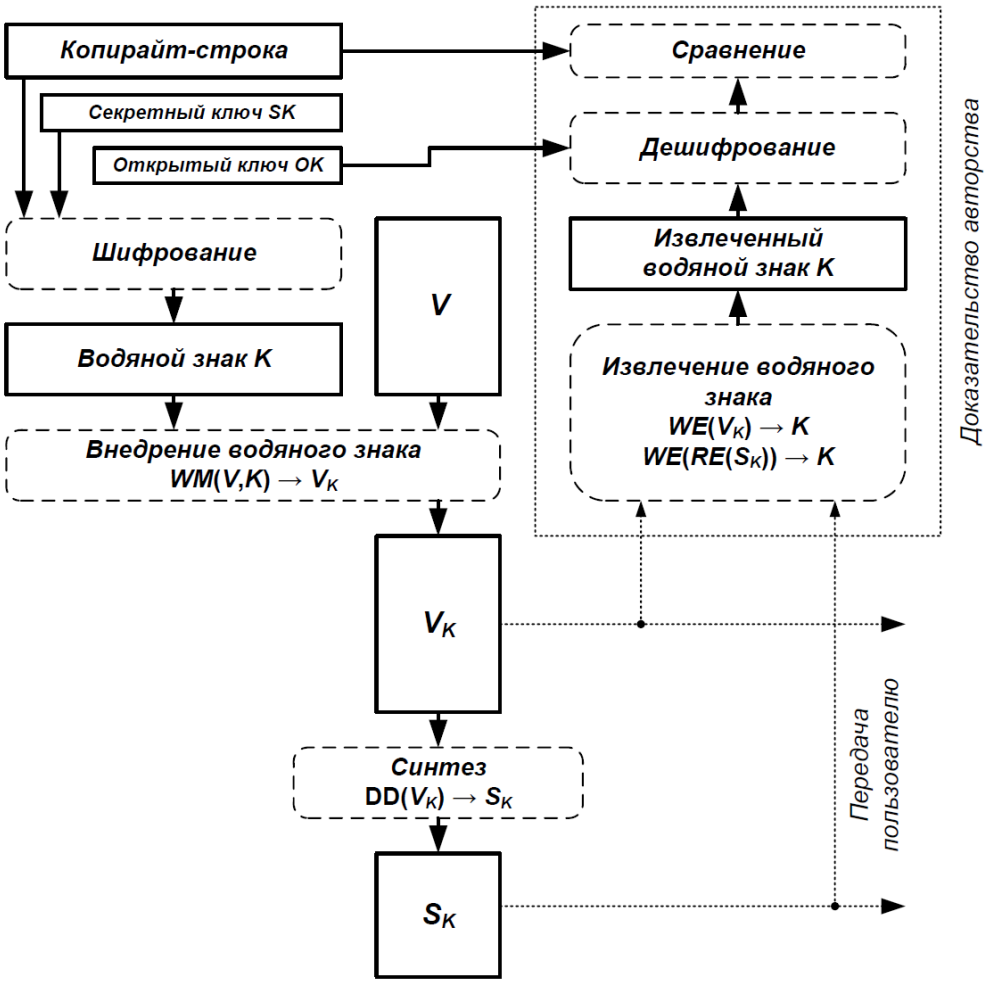


Рисунок 2.2 Обобщенная схема внедрения «водяного знака»

Однако большинство серийно выпускаемых ПЛИС не содержат регистров уникальных идентификаторов, что затрудняет решение вышеперечисленных задач разработчиками цифровых систем. В свою очередь пользовательская реализация соответствующих регистров ресурсами ПЛИС не защищена от клонирования (несанкционированного повторения и использования) как во время создания проектных описаний, например исходных HDL-кодов, так и после реализации в аппаратуре.

Одним из самых распространенных видов атак на цифровые

устройства, которые реализованы на ПЛИС типа FPGA, является клонирование BIT-образа. В силу того, что исходные проектные описания (netlist, VHDL-коды) и сами BIT-образы конфигурации FPGA представляются в открытом формате, злоумышленник может без особых технических усилий нелегально использовать их и тиражировать. Помимо этого имеющийся в руках злоумышленника BIT-образ может быть подвергнут обратному проектированию для возможности получения netlist и понимания функционирования цифровой системы. Но чаще BIT-образ подвергается клонированию (например для реализации на более дешевых FPGA) в качестве «черного ящика» без анализа и изменения функционирования.

Одним из вариантов защиты цифровых систем для FPGA является применение элементов аппаратной криптографии, в частности механизма аутентификации [7]. На рис. 3.3 представлена обобщенная схема аутентификации вида «запрос—ответ» цифрового устройства, реализованного на FPGA.

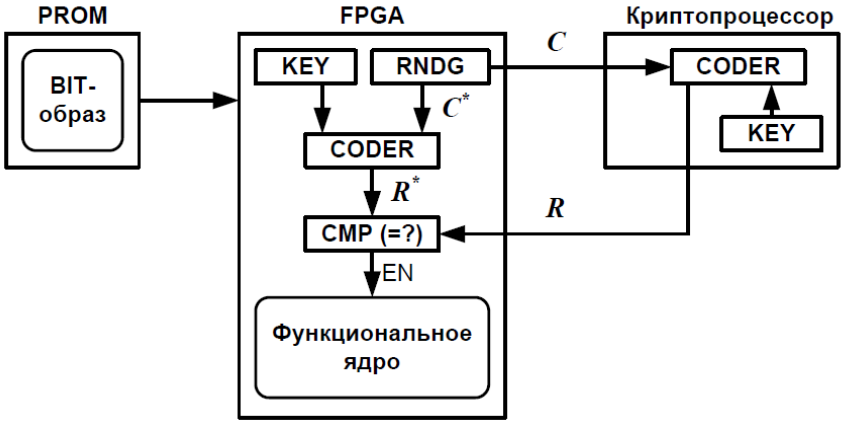


Рисунок 2.3 Обобщенная схема аутентификации вида «запрос—ответ»

Проектируемое цифровое устройство, помимо функционального ядра, содержит аппаратные блоки схемы аутентификации: регистр секретного ключа KEY, генератор случайных чисел RNDG, шифратор CODER и схему

сравнения на равенство CMP. В состав всей цифровой системы, кроме FPGA, включены СБИС энергонезависимой памяти PROM, которая хранит BIT-образ реализуемого устройства, и криптопроцессор, обеспечивающий реализацию криптографических алгоритмов, в том числе и алгоритма аутентификации. Необходимым условием реализации аутентификации является наличие копии секретного ключа KEY и схемы шифрования CODER на стороне криптопроцессора. В подобных системах наиболее уязвимым является сам секретный ключ KEY. С точки зрения злоумышленника наиболее предпочтительным выглядит процесс обратного проектирования BIT-образа EPGA для определения значения KEY. В этом случае желательно подвергать обфускации не только исходные HDL-описания цифрового устройства, но и аппаратную структуру схем аутентификации (в частности, регистра секретного ключа).

При инициализации системы BIT-образ, хранящийся в открытом виде, передается по открытому каналу для конфигурации FPGA. В начальный момент времени функционирования реализованного устройства генератор RNDG передает криптопроцессору по открытому каналу запрос C. На стороне PPOA копия сгенерированного запроса C* используется для шифрования секретного ключа KEY и получения значения B*. В это же время криптопроцессор осуществляет аналогичные действия, оперируя исходным значением C и копией ключа KEY. Полученный результат R передается по открытому каналу на сторону FPGA для сравнения со значением R*. В случае равенства значений (R = R") принимается решение о разрешении функционирования основного ядра цифрового устройства. В противном случае функционирование системы блокируется. К недостаткам такого подхода можно отнести наличие дополнительной аппаратуры в виде криптопроцессора, хранение секретного ключа в исходных проектных описаниях и в BIT-образе, наличие дополнительной аппаратуры, блокирующей функционирование защищаемого устройства.

Более надежной является методика, основанная на шифровании BIT-образа FPGA представленной на рисунке 2.4.

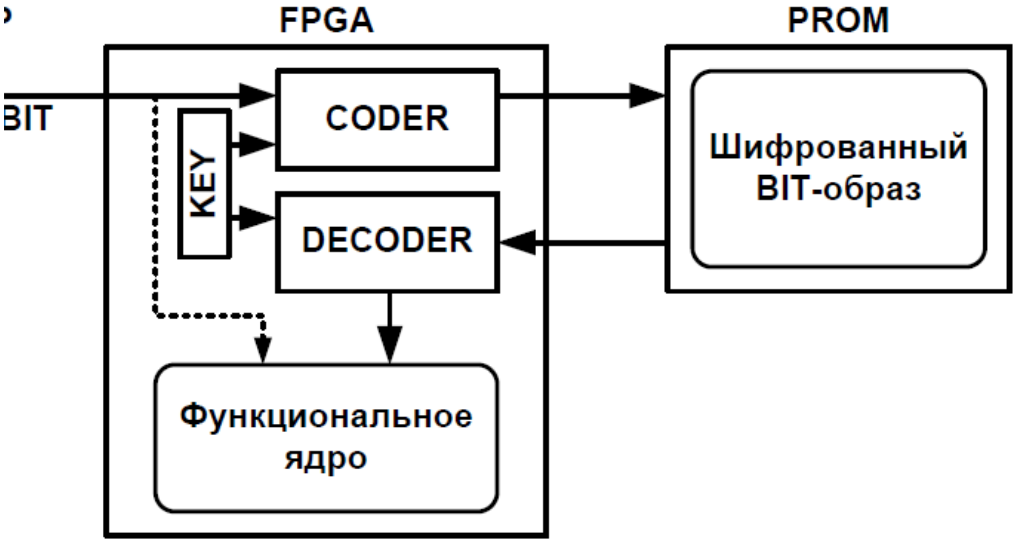


Рисунок 2.4 Схема аутентификации

Данная методика основана на использовании аппаратного шифрования и дешифрования BIT-образа на стороне СБИС FPGA, при этом секретный ключ шифрования также находится в FPGA и может представлять собой уникальный идентификатор, значение которого однократно программируется при изготовлении либо может быть переопределено пользователем.

Полученный при помощи средств автоматизированного проектирования BIT-образ передается по открытому каналу от рабочей станции проектировщика непосредственно в FPGA. Для передачи сконфигурированной цифровой системы сторонним пользователям проектировщик инициализирует PROM BIT-образом, который подвергается шифрованию посредством блока CODER и секретного ключа KEY. В начальный момент использования системы шифрованный BIT-образ по открытому каналу передается на сторону FPGA, где предварительно дешифруется посредством блока DOECODER и ключа KEY, и только после этого используется для конфигурации FPGA.

Проблема защиты от клонирования цифровых устройств на FPGA, не

содержащих описанных выше аппаратных ресурсов, остается актуальной. Для решения задачи идентификации цифрового устройства проектировщики используют уникальные идентификаторы других интегральных схем, входящих совместно с ПЛИС в состав цифровой системы.

Проектировщик, перед передачей пользователю сконфигурированной цифровой системы, вычисляет значение сигнатуры идентификатора Flash памяти, например посредством использования hesh-функций: HESH(ID) — > IDh. Полученное значение внедряется в проектное описание цифрового устройства, например в качестве «отпечатка пальца». В состав реализованного устройства входят следующие функциональные блоки: блок чтения уникального идентификатора ID, блок извлечения «отпечатка пальца» IDh, блок аппаратной реализации hesh-функции и блок сравнения на равенство. Данная схема представлена на рис. 2.5.

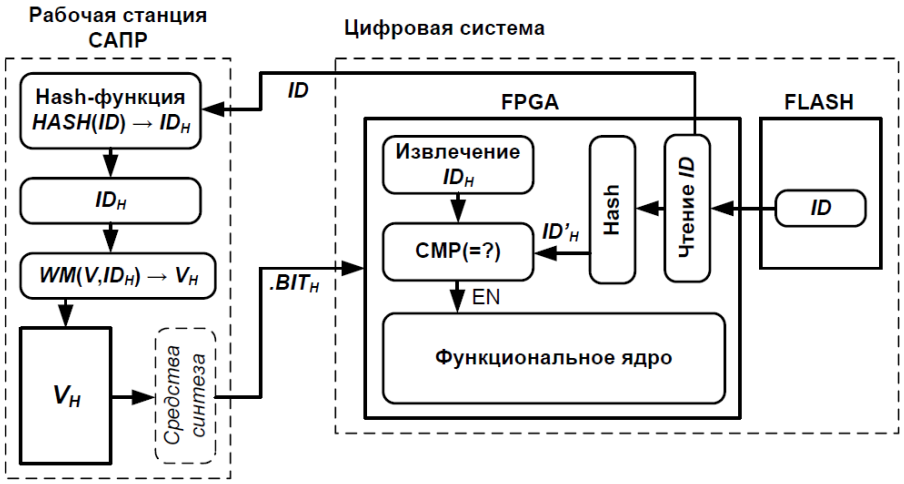


Рисунок 2.5 Идентификация посредством Flash памяти.

Проектировщик, используя значение уникального идентификатора Flash-памяти, получает его hesh-сигнатуру, значение которой внедряет в проектное описание: WM(V,IDh) — > Vh. Далее описание Vh подвергается синтезу с генерированием образа конфигурации FPGA BITh.

При инициализации цифровой системы сначала осуществляется чтение значения идентификатора ID из Flash-памяти и извлечение ID из ресурсов

реализованного цифрового устройства. Затем блок аппаратной реализации hesh-функции вычисляет значение ID'h, которое сравнивается с извлеченным Idh. В случает равенства двух значений принимается решение о разрешении функционирования основного ядра цифрового устройства.

Таким образом, идентификация цифрового устройства, реализованного на FPGA, осуществляется при помощи уникального идентификатора Flash-памяти, что затрудняет возможность несанкционированного клонирования как BIT-образа, так и всей цифровой системы в целом.

2.5. Методы физической криптографии для цифровых устройств на ПЛИС

Физическая криптография, основанная на структурной сложности электронных систем, находит свое применение не только в методах защиты конфиденциальной информации, но все чаще применяется для обеспечения защиты цифровых систем от нелегального использования. Впервые идею применения физически неклонируемых функций PUF (Physical Unclonation Function) для цифровых систем предложил Р. Паппу (R. Pappu) в своих работах [11, 12], а современное толкование и формальное определение PUFP было предложено П. Туилсом в 2007 году [13]. По определению Туилса физически неклонируемой функцией является характеристика физической (цифровой) системы, которая не подлежит клонированию (копированию, воспроизведению) на других системах. Цифровые системы состоят, как правило, из множества компонент, физические параметры которых на стадии производства принимают случайные значения. Подобное свойство получило название физической вариации технологического процесса. В процессе создания цифровых систем принципиально невозможно управлять величинами физических параметров их компонент, определяя для них конкретные значения. Таким образом, наличие подобных случайных параметров делает каждую цифровую систему уникальной и физически

невоспроизводимой (неклонируемой). Идея извлечения уникальных параметров из цифровых систем лежит в основе аппаратных PUF.

Формально PUF описывается значениями пар входных и соответствующих им выходных параметров, которые для аппаратных PUF являются соответственно значениями входных сигналов запроса C и значениями выходных сигналов ответа R. Сама же PUF является функцией преобразования множества запросов Ci во множество ответов Ri: Ri = PUF(Ci)

В 2009 году У. Рурмаир [114] дал общее определение PUF в представлении систем со сверхбольшим объемом информации. По определению У. Рурмаира физически неклонируемые функции представляют собой сложные неуправляемые физические системы со сверхбольшим объемом структурной информации, которые удовлетворяют следующим свойствам.

1. Информация из таких систем может быть неоднократно извлечена с высокой степенью надежности путем подачи различных запросов Ci, и получения множества ответов Ri;.

2. Количество возможных запросов Ci должно быть достаточно велико, чтобы все возможные значения соответствующих ответов Ri, не могли быть получены путем полного перебора всех возможных запросов Ci за приемлемый временной отрезок времени.

3. Обладая информацией о паре запросответ {Ci, Ri}, нет никакой возможности рассчитать, смоделировать либо каким-то другим математическим способом предсказать значение пары {Cj, Rj}, или другое множество таких пар.

4. Для физической системы со сверхбольшим объемом структурной информации должно быть чрезвычайно сложно (практически не возможно) ее физическое воспроизведение либо клонирование как аналогичной системы, имеющей идентичное множество пар {Ci, Ri}.

Для цифровых устройств идея создания PUF основана на использовании физических вариаций технологического процесса создания интегральных схем. Такие вариации носят случайный характер и не могут быть предсказаны, а тем более клонированы. Незначительные отклонения физических параметров при изготовлении идентичных по функциональности интегральных схем в первую очередь выражаются в различии их параметрических характеристик, например в задержках распространения сигналов [15].

Таким образом, задача реализации аппаратных PUF может быть сформулирована как создание цифрового устройства, позволяющего принимать по своим входным портам множество запросов Ci,- и вырабатывать на своих выходных портах множество ответов Ri таким образом, чтобы пары {Ci, Ri} были уникальными, непредсказуемыми и неклонируемыми на других идентичных по функциональности и топологии интегральных схемах.

Первые примеры реализации аппаратных PUF были основаны на измерении задержки распространения сигналов в реконфигурируемых путях цифровых устройств. Под «путем» цифрового устройства понимают множество последовательно подключенных логических элементов, которые позволяют транслировать логическое значение выбранного сигнала. Каждый элемент в пути характеризуется задержкой d<1 распространения сигнала.

В свою очередь величина параметра d определяется двумя составляющими: d = ds + dr, ds >> dr, где ds — величина статической задержки элемента, dr — случайная динамическая компонента, зависящая от физических вариаций технологического процесса изготовления цифрового устройства.

Число элементов пути определяет его длину и значение суммарной задержки распространения сигнала от начала пути до выхода последнего элемента. Чем длиннее путь, тем больше результирующая случайная

динамическая составляющая dr, которую можно зарегистрировать (измерить) посредством цифровой логики. Значение результирующей составляющей dr в принципе не может быть одинаковым на различных кристаллах интегральных схем. Одним из наиболее известных методов реализации аппаратных PUF, основанных на измерении задержек распространения сигналов, является PUF типа арбитр.

2.6 PUF типа арбитр.

Основная идея реализации PUF типа арбитр лежит в построении двух топологически и функционально идентичных путей на одном кристалле интегральной схемы. Такие пути носят название «симметричные пути», и имеют очень близкие по значениям величины времени распространения сигналов по ним. Однако физически такие пути являются принципиально различными благодаря различию результирующих составляющих dr для каждого из них. Измерение различий во времени распространения сигналов по симметричным путям (измерение длин путей) может быть осуществлено посредством одновременной подачи на входы обоих путей фронта сигнала и определении на выходах, какой путь оказался длиннее. Пути проектируются в виде конфигурируемых симметричных соединений при помощи мультиплексоров, количество которых определяет не только протяженность симметричных путей, но и мощность множества путей, из которого возможна выборка двух путей для дальнейшего сравнения. По сути, управляющие входы мультиплексоров принимают множество значений {0.1}, которые могут восприниматься в качестве запроса Ci значение которого определяет конкретную пару путей. Одной из простейших схем сравнения (арбитр) симметричных путей является синхронный D-триггер, на вход данных которого подается сигнал с выхода первого пути, а на вход синхронизации — сигнал с выхода второго пути. Таким образом, формируемое значение на выходе триггера {0.1} является результатом сравнения двух выбранных путей, а с точки зрения неклонируемой функции

на выходе арбитра формируется значение ответа Ri;. Типовая конфигурация PUF типа арбитр приведена на рисунке 2.6.

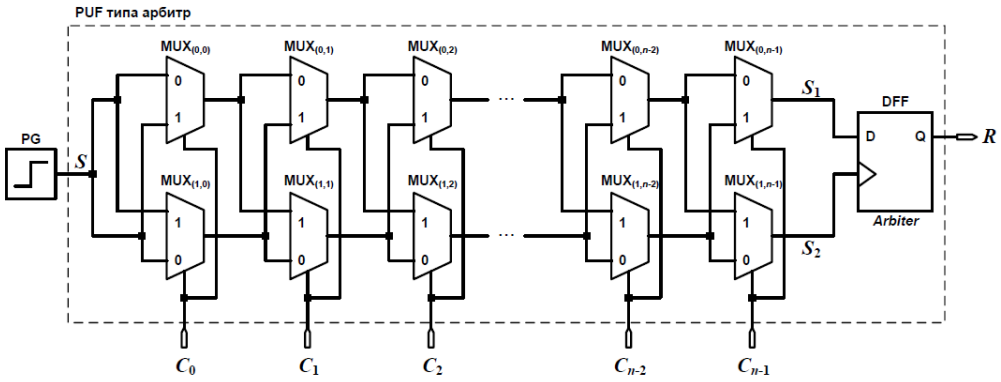


Рис 2.6 Типовая конфигурация PUF типа арбитр

Схема имеет один входной порт цифрового импульса S, вырабатываемого генератором PG, n входных портов сигналов запроса Ci; (Co, C_1 ..., C_{n_1}) и один выходной порт ответа R. В состав схемы включены 2n двухвходовых мультиплексоров MUX(i,j) (i {0.1}. ; j {0,.....n —1}) и синхронный D-триггер DFF. Множество мультиплексоров формирует конфигурируемые цифровые пути, звеном которых является пара двух смежных мультиплексоров MUX(0,j) и MUX(1,j). Выходы всех звеньев, за исключением последнего, перекрестно подключены к нулевым и единичным входам мультиплексоров последующих звеньев. При нулевом значении управляющего сигнала Ci = 0 мультиплексор MUX(i,j) коммутирует сигнал со своего нулевого входа на нулевой вход мультиплексора последующего звена MUX(1,j). В случае если Ci = 1, происходит перекрестная коммутация сигналов, при которой сигналы с единичных входов мультиплексоров MUX(i,j) подаются на единичные входы мультиплексоров MUX(i,j+1).

Таким образом, для данной схемы существует возможность сформировать 2^n различных пар симметричных путей прохождения сигнала S.

Сигналы S1 и S2, значения которых формируются на выходах последней пары мультиплексоров MUX(0,n-1) и MUX(1,n-1), являются выходами двух симметричных путей, которые поступают на вход данных D

и вход синхронизации арбитра DFF. В случае если путь, который прошел сигнал S1 является длиннее пути, который прошел сигнал S2, то арбитр сформирует на своем выходе Q значение 0. В противном случае на выходе Q сформируется значение 1. Выход арбитра непосредственно подключен к выходу ответов R представленной схемы PUF

2.7 PUF на основе кольцевых генераторов

Существенно большей надежностью обладают PUF на основе кольцевых генераторов (RO-PUF, Ring Oscillator PUF) [18, 19]. Данный тип физически неклонируемых функций основан на использовании частотных характеристик кольцевых генераторов RO, представляющих собой схемы из последовательно подключенных инверторов с отрицательной обратной связью. Число инверторов должно быть нечетным, что обеспечивает формирование выходного сигнала в виде меандра, частота которого определяется величиной задержки распространения сигнала по цепи обратной связи. Для управляемости схем RO в цепь обратной связи добавляют двухвходой логический элемент И, позволяющий удерживать генератор в стабильном состоянии. В силу вариаций задержек сигнала на элементах генератора два идентичных по топологии и функциональности RO имеют принципиально различные частоты формирования выходного сигнала. Результат сравнения частот двух генераторов является основой для формирования однобитного ответа для схемы RO-PUF. Таким образом, пара генераторов RO, реализованная на одной либо двух интегральных схемах, будет иметь произвольное соотношение частот и уникально характеризовать данную пару или интегральную схему.

На рисунке 2.7 представлена обобщенная функциональная схема RO-PUF.

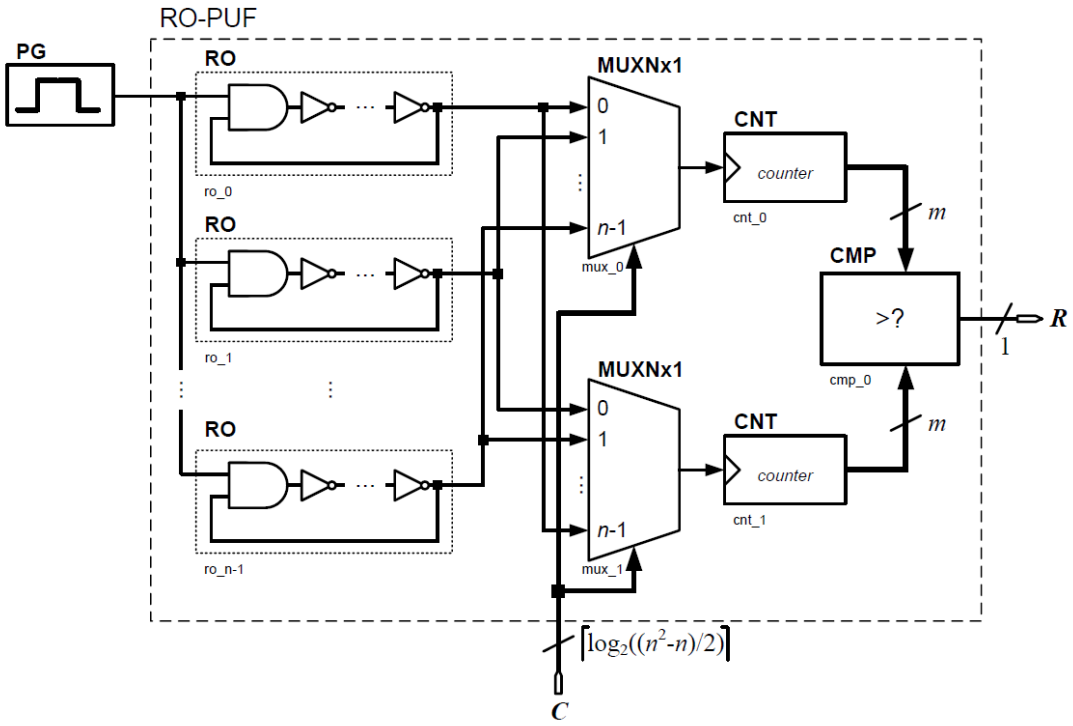


Рис. 2.7 Обобщенная функциональная схема RO-PUF.

Схема строится на n кольцевых генераторах ro_i, управление которыми осуществляется внешним источником одиночного импульса PG. Выработанные генераторами импульсы поступают на входы мультиплексоров mux_0 и mux_1, каждый из которых коммутирует один из n поступивших сигналов на свой выходной порт.

Выборка коммутируемого импульса осуществляется посредством единого селективного [log((n^2-n)/2)]-разрядного запроса C. Выходы мультиплексоров подключены ко входным портам сигнала синхронизации двоичных т-разрядных счетчиков cnt_0 и cnt_1, осуществляющих счет числа сгенерированных импульсов. По окончании счета два m-разрядных значения счетчиков подвергаются сравнению на компараторе cnt_0. Сигнал, формируемый компаратором, является однобитным ответом R на запрос C всей схемы RO-PUF.

Временной интервал измерения числа сгенерированных импульсов задается продолжительностью удержания выходного сигнала генератора PG

в значении 1, а бит ответа R сигнализирует о неравенстве частот двух выбранных генераторов RO.

2.8 PUF на основе статического ОЗУ (SRAM PUR)

Статические оперативные запоминающие устройства (СОЗУ) широко применяются в вычислительной технике для хранения данных. Элемент СОЗУ, позволяющий хранить один бит информации, как правило, строится на основе четырех транзисторов, которые реализуют два инвертора с перекрестными обратными связями (элемент с двумя устойчивыми состояниями). В цифровой схемотехнике подобным элементом может быть RS-защелка, проектируемая на логических элементах 2И-НЕ. .

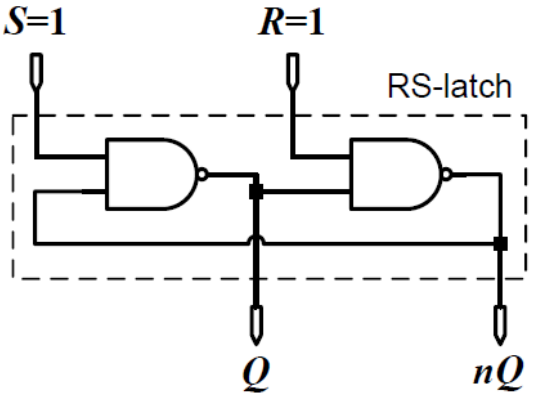


Рис. 2.8 Обобщенная схема PUF на основе статического ОЗУ

Одновременная подача значения логической единицы на входы R и S позволяет фиксировать текущее состояние защелки, которое было изменено последней операцией записи. Предположим, что такое состояние входных портов имеет место при включении питающего напряжения. В силу симметрии RS-защелки априорно неизвестно, какое конкретное значение будет зафиксировано на выходе Q(nQ). Это значение является случайным и определяется многими факторами [12].

Экспериментально подтверждено, что большинство ячеек СОЗУ при включении питающего напряжения преимущественно переходят в одно из двух возможных состояний. Происходит это в силу того, что каждый запоминающий элемент СОЗУ, являющийся RS-защелкой, имеет множество

неоднородных элементов (физически различная протяженность проводящих линий, неоднородность физических и химических свойств отдельных участков кремниевого полупроводника и т.п.).

Подобный дисбаланс является причиной того, что некоторые запоминающие элементы СОЗУ чаще переключаются в одно конкретное состояние, нежели в противоположное. Однако предсказать такое состояние практически невозможно. И только меньшая часть запоминающих элементов СОЗУ при включении питающего напряжения переключается в истинно случайное состояние (приближается к равномерному распределению), а большая часть элементов устойчиво переключается в состояние 0 либо 1.

Полученные результаты позволяют сделать вывод, что SRAM PUR являются весьма ненадежными, а особенно — применительно к программируемым логическим устройствам. Объясняется это регулярностью топологии FPGA в любом функционально симметричном элементе, например в RS-защелке, реализованной на FPGA практически всегда присутствует прогнозируемая асимметрия. Для решения этой проблемы была предложена новая концепция на базе SRAM PUR, получившая название PUF типа бабочка.

Впервые физически неклонируемая функция типа бабочка была описана в работе [14]. Основная идея реализации данной PUF основывается на эффекте SRAM PUR при включении питания со следующим изменением: одни запоминающий элемент строится на двух 0-защелках с перекрестными обратными связями, позволяющими переключать всю схему в состояние функционирования кольцевого генератора.

Выводы к разделу 2.

В разделе исследованы методы защиты цифровых проектов и аппаратных устройств на ПЛИС. Особое внимание уделено анализу модели взаимодействия пользователя и цифрового устройства, применению запутывающих преобразований HDL-описаний, анализу методик внедрение «водяных знаков» и «отпечатков пальцев» в HDL-описания.

Детально исследованы методы и способы аутентификации и идентификации цифровых устройств, на ПЛИС и физической криптографии для цифровых устройств на ПЛИС.

Иследованы физически неклонируемых функции (PUF) типа арбитр, на основе кольцевых генераторов и на основе статического ОЗУ (SRAM PUR)

3. РАЗРАБОТКА СПОСОБА ПРОЕКТИРОВАНИЯ ЗАЩИТЫ УСТРОЙСТВ НА ПЛИС

Защита цифровых проектов и устройств является естественной потребностью при создании сложных систем, которые должны обладать высокой надежностью, гибкостью и адаптацией к решаемым задачам.

Исследования комплекса прикладных задач анализа, синтеза и оптимизации системного и логического проектирования реконфигурируемых устройств [21 - 23] показали, что сдерживающим фактором построения защищенных цифровых ВС является отсутствие системы формальных математических моделей, которые учитывали бы характеристики особенности их функционирования, определяемые особенностями новой элементной базой (в частности, ПЛИС).

Таким образом, задача построения формализованной системы согласованных математических понятий, адекватно описывающих характерные особенности функционирования ВС, является актуальной и важной для современного этапа развития теории проектирования вычислительных устройств.

Принцип защищенности, положенный в основу проектирования ВС, прошел определенную эволюцию и переосмысление, что, в первую очередь, связано с прогресом элементной базы. Слово реконфигурируемый трактуется двояко: есть функциональные элементы структуры, которые объединяются тем или иным способом, при помощи каналов связи, для решения конкретной задачи, либо в системе для каждой конкретной задачи используется новая структура с новыми функциональными элементами. Такая форма реконфигурации выражается в том, что для реализации различных процессов возникают структуры в произвольном порядке.

В [21] рассмотрена классификация и проанализированы основные особенности ВС, которые в наибольшей мере определяют способ их функционирования:

1. Отсутствие заранее известной структуры решения задачи;

2. Отсутствие единого событийного времени;

3. Внутренний и внешний недетерменизм при организации вычислительного процесса;

4. Наличие конфликтов и совместно используемых ресурсов;

5 Независимость структурной организации от времени;

6 Наличие двух видов рекофигурирования.

Ни одна из известных моделей функционирования РВС [22] [23] и изложенная в [24] не учитывает все эти особенности РВС в полной мере. Упомянутые модели создавались с целью спецификации структуры и исследования вычислительных процессов при решении задач в ней. Основные недостатки указанных моделей - это отсутствие связи с количественным и алгоритмическим анализом вычислительных устройств. Модели частично описывают влияние функциональных элементов структуры на динамику взаимодействия структурных элементов системы. Они не содержит множественного времени как количественной сущности, что затрудняет описание процесов временной реконфигурации. В известных моделях отсутствует иерархичность и структуризация функциональной организации вычислительной системы с точки зрения ее реконфигурируемости.

Таким образом, проанализированные выше модели лишь частично формализуют процесс описания РВУ и требуют дополнительных инструментов для логического и системного проектирования вычислительных устройств.

Способ проектирования защищенности структуры, которая достигается путем разработки согласованной системы математических понятий, адекватно описывающих их характерные особенности.

Разработка модели функционирования реконфигурируемых вычислительных устройств.

Основу модели функционирования защищенных реконфигурируемых вычислительных устройств, которые учитывают влияние функциональных элементов структуры на динамику взаимодействия структурных элементов, составляют три понятия:

1. Поведение – модель динамики организации реконфигурируемых вычислительных процессов;

2. Среда – модель аппаратных средств с реконфигурируемой структурой;

3. Реализация – исчисление, определяющее интерпретацию поведения в конкретной среде при заданных исходных данных.

Основные понятия и определения предлагаемой модели.

В составе РВУ будем выделять три основные части:

- физическую среду – аппаратные средства, определяемые составом функциональных элементов и коммутирующей сетью;
- логическую среду – управляющие средства, которые поддерживают работоспособность физической среды и реализуют выполнение той или иной задачи и способ реконфигурации;
- поведенческую среду, которая определяет конкретную реализацию вычислительного процесса на РВУ.

В предлагаемой модели под словом реконфигурация понимается децентрализация управления и вычислений в физической и логической средах. Это значит, что в физической среде есть устройства способные функционировать автономно, т.е. решать индивидуальную “перегружаемую” задачу, а логическая среда обеспечивает использование реконфигурируемости физической среды. В качестве таких устройств мы будем иметь ввиду ПЛИС. Такое понимание охватывает широкий класс вычислительных систем, к которым относятся: измерительно-вычислительные системы, бортовые вычислительные комплексы, системы управления экспериментом, технологическими процессами,

высокопроизводительные вычислительные комплексы на базе спецпроцессоров.

Динамику функционирования РВУ определяет взаимодействие процессов реализации конкретной вычислительной подзадачи при ее решении и совместная работа с логической и физической средами. Схематично наше представление о динамике можно охарактеризовать так. Каждый отдельно взятый вычислительный процесс определяет логическую последовательность действий. Эти действия есть либо реализация статического, либо динамическое использование логического ресурса. Использование сопровождается передачей параметров. Так выглядит выполнение совокупности процессов в РВУ с точки зрения глобальной организации вычислительного процесса.

Результатом выполнения действия по отношению к логической среде является формирование сообщения определенного типа и имени процесса, которому необходимо передать управление и само сообщение. Под сообщением мы будем понимать совокупность параметров обращения. Тип сообщения – это класс эквивалентности на множестве допустимых сообщений.

Передача управления может происходить по инициативе самого процесса, а может произойти и в результате воздействия на исполнителя извне (прерывания), например со стороны другого исполнителя под влиянием выполняемого на нем процесса. Одному исполнителю может быть сопоставлено несколько процессов. Их количество ограничено физическими характеристиками исполнителя.

Исполнитель – это модель физической среды. Исполнители бывают динамические и статические. Статический исполнитель может выполнять только фиксированные задачи из заранее определенного конкретного набора. Динамический может менять набор задач. Исполнители, объединенные каналами связи, могут обмениваться сообщениями и образуют

реконфигурируемый исполнитель. Канал передает эти сообщения без каких либо изменений от одного исполнителя к другому, внося определенную временную задержку. Величина этой задержки и объем единовременно передаваемой информации есть постоянные характеристики данного канала.

Каждый реконфигурируемый исполнитель связан с определенным набором исполнителей. Этот набор ограничен и не изменяется в процессе реализации задачи. Последовательность логических действий по реализации процесса также фиксирована.

Так кратко, на содержательном уровне, можно описать наше представление о существе рассматриваемого явления и те концепции, на которых создается модель. Подчеркнем, что мы задаем модель не какой то конкретной вычислительной системы, а модель класса реконфигурируемых вычислительных устройств, характеристики которого были сформулированы в [21].

Математическая модель функционирования реконфигурируемых вычислительных устройств.

Конкретизируем понятие поведения, т.е. детально опишем модель динамики организации реконфигурирования вычислительного процесса.

Поведение процесса будем описывать в форме тензорного уравнения как умножение со сверткой подмножества области отправления отображения на график этого отображения. Результатом при этом, является подмножество области прибытия отображения. [25]

Использование такого описания поведения процесса позволяет отказаться от меток шагов преобразования, т.к. их роль представляют обозначения подмножеств областей отправления. Более того, в этом случае отпадает необходимость в описании момента передачи управления при описании поведения процесса, т.к. каждое отображение начинает выполняться только тогда, когда в результате выполнения других отображений сформируется подмножество области отправления данного

отображения.[26].

Таким образом, поведение процесса можно описать тензорным уравнением преобразования его шагов, что позволяет рассматривать множество взаимодействий реализации алгоритмов конкретной задачи.

Пусть M – множество типов сообщений в РВУ и P – множество наблюдаемых процессов в системе. Множество P состоит из элементов, которые принадлежат либо множеству процессов логической среды P_L, либо множеству подзадач вычислительного алгоритма P_A, причем P = P_L ∪ P_A и P_L ∩ P_A = 0.

Выполнение шага процесса Pi определяется следующим образом:

1. инициализируется воздействие, которое заключается в получении сообщения типа a из множества M_I и управления от процесса P_j;
2. выполняется реализация, которая состоит из последовательности внутренних действий $\{q_i\}$, множество которых обозначим через Q, направленных на определение отклика на инициализируемое воздействие.
3. формируется реакция, которая заключается в посылке нового сообщения b из множества M_O.

Шаг процесса S определим как тройку S = (M_I, M_O, P), где M_I - множество исходных сообщений, M_O - множество выходных сообщений (M = M_I ∪ M_O), P – множество состояний наблюдаемых процессов.

В этом случае, выполнение шага описывается тензорным уравнением

$$S^{M\_O,P_j} = S^{M\_I,P_i} Q^{M\_O,P_j}_{M\_I,P_i} \qquad (1).$$

Важной характеристикой шага является его сложность. Сложность шага – величина, определяющая время выполнения его внутренних действий. Эти действия суть использования статических логических ресурсов, поэтому можно определить тензор W_i^{qi}, который однозначно задает сложность $W_i^{qi} = W_1^{q1} + W_2^{q2} + \ldots = W^q$. Очевидно, что если внутренние действия отсутствуют, то $W_i^{qi} = 0$.

Для оценки времени на передачу сообщений между исполнителями мы введем такую характеристику сообщения, как временная сложность шага. Определим ее с помощью тензорного уравнения $C\_T^b = M\_I_a^i Q_i^{ab}$, которое реализует график M_O^b на множестве типов сообщений с областью значений – множество натуральных чисел *i*.

Историей процесса P назовем непустую конечную последовательность его шагов, замкнутую слева, т.е. если $H_P S^P_i = S_1\ S_{2....}\ S_k$ история, то $\forall i : i \le k$ и последовательности $S_1\ S_{2....}\ S_k$ – история. Наше предположение о конечности истории основано на представлении о реализуемости вычислений под надзором наблюдателя, а всякое наблюдение конечно.

Для выполнения процесса P потребуется фиксированное число шагов его истории, которые можно охарактеризовать длиной истории K_P. Истории процессов P_1 и P_2 равны тогда и только тогда, когда равны их длины $K_{P1} = K_{P2}$ и $\forall i : 1 \le i \le K_{P1}$, $S_{1i} = S_{2j}$, где S_{ij} –
i-тый шаг j-го процесса.

Два шага S_1 и S_2 равны тогда и только тогда, когда совпадают типы сообщений и типы процессов. Равенство сложностей шагов в этом случае не требуется.

Подисторией процесса P называется любой постфикс этой истории. Цепь – это любое подслово слова H_P. Поведением процесса P назовем множество всевозможных историй этого процесса, которое мы будем обозначать BH_P.

Проанализируем структуру поведения процесса BH_P с целью его описания. Для этого историю процесса $H_P S^P_i = S_1\ S_{2...}\ S_k$, правая часть которой представляет цепь $k = S_1\ S_{2...}\ S_k$ представим в блочном виде: $S_1\ S_{2...}\ S_l = l_1$, $S_{l+1}\ S_{l+2...}\ S_k = l_2$.

Введем две операции: следования $\rightarrow$ и альтернативы $\neg$. Цепь $k = l_1 \rightarrow l_2$ такова, что

$k = l_1 \neg l_2$, а l_1 – префикс, l_2 – постфикс k, т.е. операция следования есть конконтенация цепей, рассматриваемых как последовательность символов в алфавите S^P_i. Нетрудно видеть, что операция $\rightarrow$ ассоциативна, но не коммутативна.

Если поведение процесса P задается выражением $H_{P1} \neg H_{P2}$, то сам процесс, в зависимости от тех или иных условий, может быть реализован либо историей H_{P1} либо H_{P2}. Правило выбора способа реализации операция $\neg$ не определяет. Основные свойства операции $\neg$ следующие:

1. $l_1 \neg l_2 = l_2 \neg l_1$;
2. $(l_1 \neg l_2) \neg l_3 = l_1 \neg (l_2 \neg l_3)$;
3. $l_1 \rightarrow (l_2 \neg l_3) = l_1 \rightarrow l_2 \neg l_1 \rightarrow l_3$.

Таким образом, операция $\neg$ коммутативна, ассоциативна и дистрибутивна справа относительно операции $\rightarrow$.

Теорема 1. Любой процесс P, который имеет историю H_P ($P : H_P \in BH_P$) может быть представлен в виде: $H_P = H_{P1} \rightarrow \sim H_P$.

Теорема 2. Пусть BH_P' множество всевозможных историй $H_P' = H_{P0} \rightarrow \sim H_P'$, причем $\sim H_P' \in BH_P'$, где BH_P' - множество подисторий историй из BH_P'. Тогда $BH_P' = H_0 \rightarrow \Sigma \sim H_{Pi}$, где сумма берется по $\sim H_{Pi} \in \sim BH_P'$.

Обозначим элементы $t^* H_{P0}$ такими, что они не принадлежат $H_{P0} \rightarrow l$, где $l \in S^*_P$.

Теорема 3. Для любого фиксированного $H_{P0} \in BH_P$ представление $t^* H_{P0} \in BH_P$ в виде $H_0 \rightarrow \Sigma \sim H_{Pi}$ единственно с точностью до ассоциативной перестановки слагаемых.

Следствие. Префикс H_P' любой истории из BH_P задает на истории класс эквивалентности $t^* H_P' = \{H_P \mid H_P = H_P' \rightarrow \sim H_P\}$. Причем для $\forall$ H_P', H_P'' из BH_P таких, что $H_P' \neq H_P''$ следует, что $t^* H_P' \cap t^* H_P'' = \varnothing$.

Теорема 4. Для любого множества BH_P существует тензор преобразования (возможно инвариант), который устанавливает взаимно-

однозначное соответствие между множествами путей историй процесса и множества историй из $ВН_Р$:

$$S_{P_i}^{k} T_k^{P_i} = BH_{P_i}$$

Одной из основных особенностей поведения реконфигурируемых вычислительных процессов является недетерминизм. Можно определить два его вида: внутренний и внешний. Будем говорить, что выражение $S_i + S_j$ описывает внешний недетерминизм, если множество M_I из процесса S_i не равно соответствующему множеству исходных сообщений из процесса S_j и внутренний, если они равны.

Способ реализации задачи определяется множеством поведений процессов $BH_p = \bigcup_{P_i \in p} bhp_i$ образующих ее решения. Множество шагов $S^P{}_i$ в поведении $ВН_Р$ частично упорядочено. Это отношение отражает взаимодействие между процессами и порядок следования шагов в истории каждого процесса. Проанализируем это отношение более подробно, т.к. оно описывает причинно-следственные связи на множестве действий при решении задачи и во многом определяет семантику реконфигурации в РВУ. По определению, из цепочки S_i следует цепочка S_j тогда и только тогда когда

$\exists\ Н_Р \in ВН_Р : (S_i \neg S_j) \in Н_Р$ и , $S_j \in S^P{}_k$; где S_i – причина, а S_j – следствие. Это отношение обозначим R □ (P) = { (S_i, S_j) | из S_i следует S_j }. Как видно, оно означает причинно-следственные связи на $S^P{}_k$, где $k \in P$.

Это отношение можно распространить на случай событий. Выполнение шага процесса P согласно (1) состоит в инициализации воздействия путем получения сообщения из M_I и реакции, которая заключается в формировании нового сообщения из M_O. Если обозначить событие через *e,* с надлежащим индексом, то получим: из цепочки e_i следует цепочка e_j тогда и только тогда когда

$\exists\ S_k : (e_i = M\_I(S^P{}_k)\ \&\ e_j = M\_O(S^P{}_k) \vee \exists\ S_l :\ e_j \in S_k\ \&\ e_j \in S_l\ \&\ (S_k, S_l) \in R$ □ (P))

Определим отношение

$R\sim(P) = \{(S_i, S_j) \mid \exists Hp_1, \exists Hp_2 : S_i \in Hp_1 \,\&\, S_j \in Hp_2 \,\&\, M\_O(S_i^P) = M\_I(S_j^P)\}$,

т.е. реакция на шаге S_i в поведении процесса P_1 есть обращение к шагу S_j в поведении процесса P_2. Если $(S_i, S_j) \in R\sim(P)$, то этот факт будем записывать как тензорное уравнение $S_j = S_i R^i_j$. Отношение $R\sim(P)$ несимметрично, иррефлексивно.

Обозначим $R > (P) = R\sim(P) \cup \{ \bigcup_{P_i=P} R\square(P) \}$. Не трудно показать, что эти отношение также несимметрично и иррефлексивно.

По аналогии с ранее сделанным, распространим отношение $R\sim(P)$ на случай событий:

из цепочки e_i следует цепочка e_j тогда и только тогда когда

$\exists\, S_k\ \exists\, S_l : (S_k\ S_m) \in R\sim(P)\ \&\ e_i \in S_k \,\&\, e_j \in S_m$

Это отношение обозначим $Re\sim(P)$.

Историей выполнения вычисления называется тройка $(H^i_{P,}\ M\_I_i{}^j, R\sim P)$, т.е. это совокупность историй процессов H^i_P , удовлетворяющих отношению $R\sim(P)$; $M\_I_i{}^j$ – сообщение типа a_i из множества исходных сообщений процесса P^j , которое задает первый шаг в том процессе, с которого начинаются вычисления.

Рассмотрим различия в описании поведения процессов логической среды и описания вычислений, проводящихся в ней.

1. В поведении прикладных процессов вместо некоторого шага может быть указан тензор процесса, который свернут до шага. Это означает, что процесс начиная с этого места ведет себя так как свернутый процесс.

2. В уравнениях выполнения процессов из логической среды, последовательности шагов вычислений представляются переменными с именами специального фиксированного вида, которые образуют множество Dm с элементами dmi . Реакция в которой указано dmi , означает передачу управления прикладному процессу, сопоставленному этому имени. Соответствие между Dm и процессами из P устанавливается уравнением $SHD = P^i\ Dm_i$. Здесь предполагается, что SHD не зависит от времени. Однако,

наделяя SHD различными свойствами, например, вводя индексы времени, можно описывать динамические процессы развивающиеся во времени.

3. Считаем, что множества Dm и P_L фиксированы для задачи. В этом случае, исходя из ограничения Dm, следует ограничение на множество допустимых к реализации вычислений, а именно | Dm | ≥ | P |.

4. В логической среде есть процесс с именем Stop, обращение к которому завершает процесс. В реакциях шагов может присутствовать преодпределенная переменная back, которую можно представить, как вершину стека. Для работы с ней есть две функции put – поместить в стек и get – выдать значения.

Проанализируем виды реконфигурирования и их семантику. Есть два вида реконфигурации: функциональная и полная. Первая соответствует случаю, когда функциональные элементы, объединяются при помощи коммутационной сети для решения той или иной подзадачи. В этом случае, выполнение вычислений происходит одним исполнителем, т.е. процессы разделяют ресурсы одной и той же логической и физической сред. Во втором случае, вычисления выполняются независимо на разных процессах, обмениваясь при необходимости сообщениями.

Введем две операции композиции процессов: $P_1 \sim P_2$ – совместное использование функциональных элементов для решения ряда подзадач; $P_1 \parallel P_2$ – полная реконфигурация. Существует несколько подходов в описании способа реконфигурирования. Например, один из самых распространенных это использования описания типа семантики применяемой в языках программирования, таких как CSP [27]. Другой, это семантики описания операторов работы с разделяемыми переменными, который чаще всего используется в теории сетей [28].

Для описания процессов при объединении функциональных элементов с помощью коммутирующей сети будем использовать семантику чередования. Это означает, что события которые задаются процессами HP_1 и HP_2, образуют цепочку, в которой они не имеют предпочтения в порядке

следования между собой. Ограничение на чередование определяет некоторое отношение независимости, задаваемое на S^P. Такая семантика учитывает единство временной оси исполнителя. Это выражается в том, что все события, происходящие на данном исполнителе, собираются в цепочки.

Семантику независимых реконфигурируемых структур, организующих выполнение вычислительных процессов, будем описывать отношениями частичного порядка на множестве их событий. Запись $P_1 \parallel P_2$ означает, что цепочка событий, определяемая историей HP_1 , появляется между точками взаимодействия вместо и независимо от цепочки, определяемой историей HP_2. Анализ взаимодействия этих видов реконфигурации и их эквивалентность будет проведен ниже.

Конкретизируем понятие физической среды, которая в предлагаемой модели реконфигурируемых устройств представляет исполнитель. Прежде всего, рассмотрим реконфигурируемые устройства у которых функциональные элементы структуры объединяются тем или иным способом при помощи коммуникационных каналов. Для этого введем понятие пространственно реконфигурируемый исполнитель.

Пространственно реконфигурируемый исполнитель SE - это математическая структура:

$$SE=<Pa,\ Tm_n^e,\ M^P{}_i,\ C\_S,\ A_{Map}^{Tm}> \qquad (2)$$

у которой Pa – атомарный процесс; Tm_n^e -- тензор модельного времени; $M^P{}_i$ - память исполнителя; C_S – пространственно реконфигурируемый вычислительный узел; A_{Map}^{Tm} - арбитр. Определим строго эти понятия.

В множестве наблюдаемых процессов P можно выделить подмножество атомарных процессов Pa. Атомарным называется процесс, который для конкретного устройства имеет минимальное количество шагов истории процесса. В этом случае атомарный процесс задает частичное отображение множества исходных сообщений M_I в множество выходных сообщений M_O и образует множество типов атомарных сообщений. M_A.

Основные свойства атомарных процессов таковы:

1) $\forall a_i : \exists M\_I : \exists! M\_O : a_i \in M\_A$, $\forall b_i : \exists M\_O : \exists! M\_I : b_i \in M\_A$

т.е. для любого $a_i \in M\_A$ определено множество исходных сообщений множество выходных сообщений. Между M_I и M_O тензор $S^{M\_I,Pa}$ задает однозначное соответствие. Поэтому недетерминизма на уровне исполнителя нет.

2) Выполнение атомарного процесса не прерывается, т.е. начавшись, атомарный процесс будет закончен за определенное фиксированное, конечное время без останова.

3) После выполнения атомарного процесса управление всегда возвращается тому процессу, который инициировал его выполнение, если ему на вход поступило сообщение допустимого типа.

4) У атомарного процесса моменты передачи сообщения результата и моменты возврата управления совпадают; если управление передано, то сообщение доступно.

Каждый атомарный процесс характеризуется сложностью, которая определяет время выполнения его внутренних действий в тактах. Суть этих действий, заключается в использовании статических физических ресурсов сети. Поэтому можно определить тензор $\mathrm{W_i^a}$ который однозначно задает время выполнения атомарного процесса множеством натуральных единиц времени (тактов) по часам исполнителя. В этом случае можно отметить, что время выполнения любого атомарного процесса конечно, постоянно и равно $\mathrm{P_a W_i^a}$.

Модельное время есть тензор $\mathrm{Tm_n^e}$, который сопоставляет событию *e* количество тактов прошедших к моменту его возникновения на данном исполнителе. Момент наступления события считается с момента начала выполнения первого атомарного процесса. Область отправления отображения модельного времени определена на множестве всех событий e_i в процессах $\mathrm{P_a}$, приписанных данному исполнителю и представляет собой, в

общем случае, множество натуральных чисел. Постулируем свойства тензора Tm.

Постулат 1. $\forall e_i, e_j : e_i, e_j \in BHp_m \,\&\, e_i \sim\!\!\rightarrow e_j$, т.е. в одном процессе причина всегда наступает раньше следствия $Tm_i < Tm_j$.

Постулат 2. $\forall e_i, e_j : e_i \in BHp_k \,\&\, e_j \in BHp_m \,\&\, e_i \sim\!\!\rightarrow e_j \,\&\, p_m \sim p_k$, т.е. если причина и следствие принадлежат разным $Tm_i < Tm_j$ процессам и эти процессы выполняются на одном и том же исполнителе (устройстве), то по часам этого исполнителя причина наступит всегда раньше следствия.

Из приведенных выше постулатов можно сформулировать следующие следствия:

Следствие 1. Каждый процесс имеет длительность, т.е. между двумя последовательными событиями одного процесса (между причиной и следствием) всегда происходит хотя бы один такт времени.

Следствие 2. $\forall e_i, e_j : e_i \rightarrow e_j$, и $Tm_i < Tm_j$ т.е. любой процесс выполняется последовательно.

Следствие 3. $\forall e_j : e_0 \rightarrow e_j$ и $Tm_0 < Tm_j$, т.е. обращение к процессу всегда наступает раньше (по часам данного исполнителя), чем возникает какое либо событие в этом процессе.

Следствие 4. Если $\forall e_i, e_j : (e_i, e_j) \in R \sim P \,\&\, e_i \in BHp_k \,\&\, e_j \in BHp_m \,\&\, P_k \sim P_m$, то из этого следует $Tm_i < Tm_j$, причем R~P транзитивное замыкание. Данное следствие означает, что если два события связаны причинно – следственной связью, принадлежат разным процессам, которые выполняются на одном и том же исполнителе, то причина по часам этого исполнителя наступает всегда раньше следствия.

Теорема 5. Тензор Tm_n^e не нарушает отношения причинно – следственного порядка цепочки процессов, которые выполняются одним исполнителем.

Обозначим через t переменную на множестве вещественных чисел R, значение которой равно величине астрономического времени, код которого можно определить с помощью астрономических наблюдений. Такое задание времени никоим образом не зависит от свойств вычислительного устройства, но это время важно для организации конкретных вычислений.

Рассмотрим определение множества отправлений тензора Tm^{e}_{n}, а именно, обозначим через Tm_t количество тактов наступивших к моменту времени t. При этом постулируем следующие свойства:

Постулат 3. $\forall t, \exists P, \forall \Delta t : \rho, t, \ \Delta \in R \,\&\, 0 < \Delta < \rho$

В этом случае Tm_t(t + Δt) – Tm_t = 1& Tm_t(t + Δt) – Tm_t = 0. т.е. часы идут с минимально различимым тактом ρ. На практике точность измерения ρ флюктуирует. Поэтому для практического применения Постулат 3 удобнее представить в следующем виде:

Постулат 3_1. $\forall t, \forall \Delta t \forall \rho, \exists \zeta : 0 < \Delta < (\rho - \zeta) \,\&\, \rho - \zeta < \rho < \rho + \zeta$

В этом случае Tm_t(t + Δt) – Tm_t = 0& Tm_t(t + ρ) – Tm_t = 1.

При таком подходе можно однозначно обеспечить показания часов в разных экспериментах только лишь в рамках определенного интервала наблюдений δt, такого, что $|[\frac{\delta t}{\rho - \zeta}|]$ = $|[\frac{\delta t}{\rho + \zeta}|]$, где $|[\ldots]|$ - целая часть выражения. Отсюда, зная ρ и ζ можно вычислить δt, либо зная заранее δt, определить длительность шага моделирования.

Для хранения и правильного функционирования процесса требуется определенный объем памяти в котором хранятся параметры реконфигурации . Определим тензор M^{P}_{i}, который в качестве области отправления отображения на график этого отображения имеет множество процессов и множество натуральных чисел соответственно. Такое задание тензора M^{P}_{i} позволяет определить необходимый объем памяти для хранения и выполнения процесса. В тоже время сам исполнитель может иметь определенное (возможно большее) количество единиц памяти N.

На данном конкретном исполнителе может выполняться только такое множество процессов, у которых наличная память N будет больше чем необходимая.

Понятие вычислительного узла является ключевым для построения модели исполнителя. Пространственно реконфигурируемый вычислительный узел C_S это математическая структура C_S = <IN, ENT_{IN}^{n}, OUT, EXT_k^{OUT}>, где IN – множество полюсов, которые мы будем называть входами, тензор ENT_{IN}^{n} - определяет назначение входов; тем самым, каждому входу ставится во взаимно – однозначное соответствующие полюсы n вычислительного узла, OUT – множество полюсов, которые мы будем называть выходами, тензор EXT_k^{OUT} - определяет назначение выходов.

Полюса $out_i \in OUT$ обеспечивают передачу воздействий на процессы, размещенные на других исполнителях. Каждому out_i по определенному правилу или закону сопоставляется атомарный процесс P_out_i . Время срабатывания полюса out_i определяется тензором $Tm_i^{P-out_i}$. Также с каждым полюсом связана функция Map(n):N $\rightarrow\{0,1\}$; Map(n) = 1, если $n \in [Tm\_t_0, Tm\_t_0 + P_{P\_out_i} \bullet Tm_i^{P-out_i}]$, в противном случае – 0. Здесь Tm_t_0 задает воздействие на P_out_i. Назовем эту функцию характеристической.

Один out_i может быть соединен с несколькими входами in_j своего исполнителя или других исполнителей. Всем этим in_j одновременно будет передано одно и тоже сообщение. Время передачи того или иного сообщения определяется характеристикой соответствующей связи. (Способ задания характеристики будет рассмотрен отдельно). Объем и тип передаваемого за одно обращение к out_i сообщения определяет атомарный процесс P_out_i.

Каждому входу in_j можно сопоставить (способ задания сопоставления будет рассмотрен отдельно) только один выход out_i своего либо другого исполнителя и один процесс из множества типов сообщений M, управление

которому будет передано, если данный вход возбужден и выиграл арбитраж. Вход возбужден, если Map(n)=1 у выхода, связанным с этим входом.

Тензор A_{Map}^{Tm} описывает процедуру арбитража входов пространственно реконфигурируемого исполнителя. Область отправления отображения это множество значений тензор Tm, а множество характеристических функций определяет график этого отображения. Таким образом, в зависимости от состояния выходов, арбитраж позволяет однозначно передать управление на вход исполнителя.

Более сложная модель временного реконфигурируемого исполнителя. Конкретизируем эту модель.

Временный реконфигурируемый исполнитель DE это математическая структура:

$$DE = < \{SE_i\}, C\_D, M\_S_M^E > \qquad (3)$$

где $\{SE_i\}$ – множество пространственно реконфигурируемых исполнителей, C_D – распределенный вычислительный узел, M_S – поток сообщений.

Понятие распределенного вычислительного узла C_D определим как гиперграф, множество вершин которого образуют полюса C_S, входящие в состав соответствующего CE_i , т.е. это математическая структура: C_D = $<\{C\_S_i\}, E, ENT_{IN}^n, EXT_k^{OUT}>$, где

$\{C\_S_i\}$ – множество реконфигурируемых вычислительных узлов, которое обозначим через V. По сути, множество вычислительных узлов, определяется следующим образом:

$V = \cup\{OUT_i \cup IN_i \mid OUT_i \in C\_S_i \,\&\, OUT_i \in C\_S_i\}$; $C\_S_i \in C\_D$;

E – множество дуг, которые мы будем обозначать ARC, и которое определятся, как

$E = \{ARC_j \mid ARC_j \in V \,\&\, \forall j \exists! OUT \in ARC_j\}$, т.е. дугу в гиперграфе образует множество полюсов, среди которых есть только один полюс типа выход;

ENT_{IN}^n - тензор разметки входов C_D : IN – вход в узел C_D, если

$\exists C\_S_j : C\_S_j \in C\_D \Rightarrow \forall ARC : IN \notin ARC \& IN \in IN_j \& IN_j \in C\_S_j$; т.е. это один из входов C_S, не использованных в дугах C_D;

EXT_k^{OUT} тензор разметки выходов C_D : OUT, если

$\exists C\_S_j : OUT_j \in OUT_j \& OUT_j \in C\_S_j \Rightarrow \forall ARC : OUT_j \notin ARC$, т.е. это один из выходов C_S, не использованных в дугах C_D;

Поток M_S формируется множеством дуг E по которым за время tt блоками m_v передается множество M сообщений. Каждому конкретному сообщению m_v_i соответствует время передачи tt_i, измеряемое в тактах исполнителя. Пару вида < m_v_i, tt_i, > будем называть пропускной способностью канала.

Модели пространственного и временного реконфигурируемых исполнителей позволяют при помощи конструктивных правил описать любую структуру вычислительного устройства. Конкретизируем и определим алгебру носителей для построения модели структуры такого устройства.

Обозначим через C множество вычислительных узлов: $C = CS \cap CD$, где CS(m,n) – множество пространственно реконфигурируемых вычислительных узлов CS, имеющих не более n входов и m выходов; CD – множество временных реконфигурируемых узлов. Для простоты изложения, без потери общности, введем следующие обозначения: C_S(i,j) – пространственно реконфигурируемый вычислительный узел с i входами и j выходами; Plug - пространственно реконфигурируемый вычислительный узел вида C_S(1,1). Будем считать, что $CD_1 = CD_2$, если:

1) $\forall C\_S_{1i} : C\_S_{1i} \in CD_1 \Rightarrow \exists C\_S_{2j} : C\_S_{2j} \in CD_2 \& C\_S_{1i} = C\_S_{2j}$;

2) $E_1 = E_2$; $ENT_{IN}^{n1} = ENT_{IN}^{n2}$; $EXT_{k1}^{OUT} = EXT_{k2}^{OUT}$;

т.е. вычислительные узлы равны с точностью до изоморфизма.

Определим на множестве вычислительных узлов: C операции объединения, слияния входов, композиции и замыкания. Везде далее считаем

заданными вычислительные узлы $C\_D_1 = <V_1, E_1, ENT_{IN}^{n1}, EXT_{k1}^{OUT}>$, $C\_D_2 = <V_2, E_2, ENT_{IN}^{n2}, EXT_{k2}^{OUT}>$, $C\_D_1, C\_D_2 \in C$.

Объединением вычислительных узлов $C\_D_1 + C\_D_2$ будем называть узел $C\_D_3 = <V_3, E_3, ENT_{IN}^{n3}, EXT_{k3}^{OUT}> \in CD$ такой, что: $V_3 = V_1 \cup V_2$; $E_3 = E_1 \cup E_2$; $n_3 = n_1+n_2$; $k_3 = k_1 + k_2$;

$ENT_{IN}^{n3(i)} = ENT_{IN}^{n1(i)}$, если $1 \le i \le n_1$, и $ENT_{IN}^{n3(i)} = ENT_{IN}^{n2(i-n_1)}$ если $n_1 \le i \le n_1 + n_2$;

$EXT_{k3}^{OUT} = EXT_{k1}^{OUT}$, если $OUT \in V_1$, и $EXT_{k3}^{OUT} = k_1 + EXT_{k2}^{OUT}$, если $OUT \in V_2$.

Рисунок 3.1 иллюстрирует эту операцию графически.

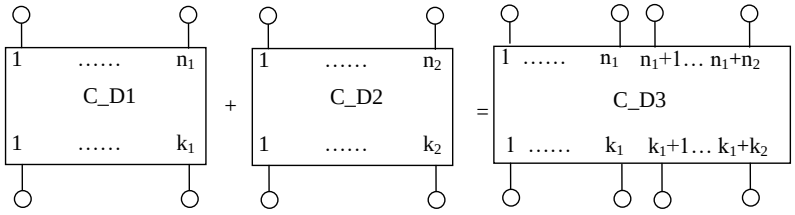


Рис 3.1.Объединение вычислительных узлов.

Слиянием входов вычислительных узлов C_D1 # C_D2 будем называть узел $C\_D_3 = <V_3, E_3, ENT_{IN}^{n3}, EXT_{k3}^{OUT}> \in CD$ такой, что: $V_3 = ENT_{IN}^{n1} \cup \{EXT_{k1}^{OUT} \cup EXT_{k2}^{OUT}\}$; $E_3 = E_1 \cup E_2$; причем $ENT_{IN}^{n2} = ENT_{IN}^{n1}$ и $EXT_{k3}^{OUT} = EXT_{k1}^{OUT}$, если $EXT_{k3}^{OUT} \in EXT_{k1}^{OUT}$, и $EXT_{k3}^{OUT} = EXT_{k2}^{OUT}$, если $EXT_{k3}^{OUT} \in EXT_{k2}^{OUT}$. Эта операция определена для тех вычислительных узлов, у которых количество входов равно друг другу, т.е. $ENT_{IN}^{n1} = ENT_{IN}^{n2}$. Сущность этой операции состоит в том, что входы C_D_1 отождествляются с входами C_D_2. Тем самым допускается многополюсное возбуждение входов узла C_D_3. Графическое представление этой операции представлено на рисунке. 3.2.

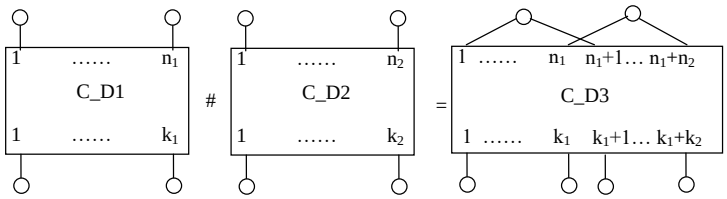


Рис. 3.2 Операция слияния структур.

Композицией носителей $C\_D_1 * C\_D_2$ будем называть вычислительный узел $C\_D_3 = <V_3, E_3, ENT_{IN}^{n3}, EXT_{k3}^{OUT}> \in CD$ такой, что: $ENT_{IN}^{n3} = ENT_{IN}^{n1}$, и $ENT_{IN}^{n3} = ENT_{IN}^{n2}$; $ENT_{IN}^{n1} = ENT_{IN}^{n3}$ $(n_1=n_3)$, $EXT_{k3}^{OUT} = EXT_{k2}^{OUT}$ $(k_3=k_2)$; $E_3 = E_1 \cup E_2 \cup (EXT_{k1}^{OUT} \cup EXT_{k1}^{OUT})$, где $k_1 = n_2$. Эта операция определена только для тех операндов, у которых число выходов первого вычислительного узла равна числу входов второго. Графически эту операцию можно проиллюстрировать рисунке 3.3.

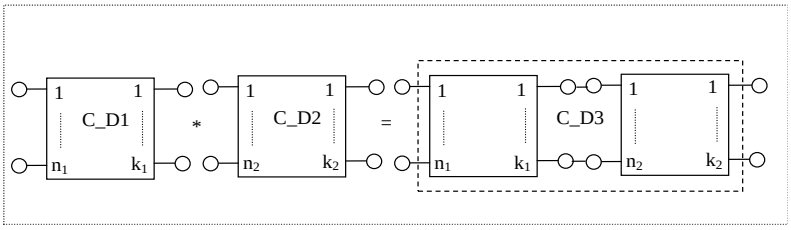


Рис 3.3 Композиция структур

Для определения операции замыкания вычислительных узлов введем предварительные определения. Обозначим через Z_{IM}^{D} тензор, который сопоставляет множеству чисел $D_g \subseteq \{1, \ldots,k\}$ график с областью значений $IM_g \subseteq \{1, \ldots,n\}$. Теперь замыканием C_D_1 по Z_{IM}^{D} называется вычислительный узел: $C\_D_2 = [C\_D_1]_{g(i)} = <V_2, E_2, ENT_{IN}^{n2}, EXT_{k2}^{OUT}>$ такой, что

$$EXT_{k2}^{OUT} = EXT_{k1}^{OUT} - EXT_{k1_{IM}}^{OUT_D} Z_{IM}^{D} \subseteq D_g,$$

$$ENT_{IN}^{n2} = ENT_{IN}^{n1} - ENT_{IN_D}^{n1_{IM}} Z_{IM}^{D} \subseteq IM_g,$$

$$E_2 = E_1 \cup \{(EXT_{ki}^{OUT}, ENT_{IN_D}^{n1_{IM}} Z_{IM}^{D}) | EXT_{k1_{IM}}^{OUT_D} \subseteq D_g\},$$

Таким образом, при выполнении операции замыкания $[C\_D_1]_{g(i)}$ вычислительного узла C_D_1 по Z_{IM}^{D} происходит перенумерация с уплотнением входов/выходов результирующего вычислительного узла C_D_2. Графически эта операция представлена на рисунке 3.4.

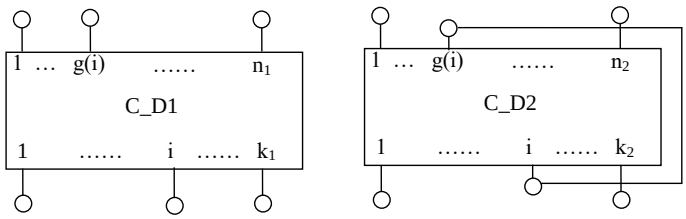


Рис. 3.4 Операция замыкания

Из выше приведенного определения вычислительного узла и операций объединений, слияний, композиции, замыкания следует, что структура < C, + , # , * , [] > образует частичную алгебру.

Теорема 6. При фиксированных m и n для любого временного реконфигурируемого вычислительного узла из CD существует разложение на множество пространственно реконфигурируемых вычислительных узлов CS с помощью операций объединений, слияний, композиции и замыкания.

Эта теорема гарантирует возможность представление структуры любого временного реконфигурируемого исполнителя в виде алгебраического выражения над пространственно реконфигурируемыми вычислительными узлами CS. Доопределим понятие исполнителя на случай алгебраического описания его структуры. В этом случае основную проблему представляет проблема переопределения потока сообщений M_S на случай вычислительных узлов типа Plug. Так как определение потока сообщений M_S на множестве дуг E определяет время tt, и блоки m_v, которыми передаются сообщения из M и задает пропускную способность канала (m_v, tt), то на случай узла типа Plug этот поток будет иметь следующий вид: $M\_S(Plug) = (m\_v, tt)_i$, если $Plug \in ARC_i$. Будем говорить, что это имеет место, т.е. $Plug \in ARC_i$, если выход узла EXT_i замкнут на вход ENT_j , такой что пара $(EXT_i , ENT_j) \in ARC_i$.

Теперь постулируем свойства тензора Tm_n^e , который сопоставляет событию *e* количество тактов n наступивших к моменту его появления на данном исполнителе на случай временного реконфигурируемого исполнителя.

Постулат 4.

$\forall\, Tm_n^e : Tm_n^e \in SE_i \in DE_i \Rightarrow (t_0 - tt) - t_0 \geq 1$, где t_0 – момент астрономического времени начала передачи единичного блока m_v (момент начала передачи – возврат управления от процесса сопоставленного EXT)

Функционирование реконфигурируемого вычислителя определяется тем, как в каждом конкретном случае будет происходить интерпретация поведения вычислительного процесса на исполнителе. Поэтому для определения способа функционирования вычислителя необходимо построить историю вычислительного процесса по исходному поведению логической среды, исполнителю и начальному воздействию.

Конкретизируем связь логической среды и исполнителя. Для этого введем тензор привязки логической среды к исполнителю $B_{DE_i}^{LE_i}$, который устанавливает взаимно – однозначное соответствие между множеством поведений процессов логической среды $LE = \{BHp_i \mid Pa_i \in \{P\_L \cup Pa\}\}$ и множеством входов / выходов исполнителя $.\{SE_i\} \in DE_i$; $EXT = \{ext_i \mid \exists\}SE : SE \in DE \& ext_i \in SE\}$; $.\{SE_i\} \in DE_i$; $ENT = \{ent_j \mid \exists\}SE : SE \in DE \& ent_j \in SE\}$.

Основные свойства такого отображения следующие: 1) одному входу соответствует один процесс; 2) нет двух или более исполнителей, которым был бы приписан процесс с одним и тем же именем; 3) тензор привязки логической среды к исполнителю $B_{DE_i}^{LE_i}$ не меняется в течении всего периода наблюдений.

Для описания функционирования реконфигурируемого вычислителя определим вычислительное устройство как тройку COM = <BHp, SHD, CE> , где BHp – поведение вычислительного процесса; SHD – значение тензорного уравнения, которое задает соответствие между множеством переменных фиксированного вида Dm и процессами; SHD т.е. это по сути функция распределения вычислительного процесса в логической среде; CE – вычислительная среда. CE = <LE, $B_{DE_i}^{LE_i}$, DE>, а свойства всех этих объектов были уже определены.

Организация вычислительного процесса по выполнению вычислений на вычислительном устройстве есть исчисление, которое будем называть временно реконфигурируемым наблюдателем. Зададим его в виде набора идентичных алгоритмов и правил взаимодействий между ними. Набор этих

алгоритмов будем называть пространственно реконфигурируемым наблюдателем (или просто наблюдателем, если это не вызывает неоднозначности). Каждый из этих алгоритмов определяет выбор очередного шага из поведения каждого из процессов порождаемых прикладными вычислениями и логической средой.

Каждому пространственно реконфигурируемому исполнителю сопоставлен свой наблюдатель, который будем обозначать OBS. По существу каждое OBS задает частичное отображение $M \times P\_A \times A \to S$, которое определяется согласно (1). В этом случае, в зависимости от шага процесса S^*, процессов P и состояния арбитра A_{Map}^{Tm} формируется следующий шаг S следующим образом:

1 Если $A_{Map}^{Tm} \neq 0$, то P = B_{Tm}^{Map} A_{Map}^{Tm} и $S \in S_i \in BHp = S_{P_i}^k T_k^{P_i}$, где S_i – очередной шаг процесса P, соответствующий воздействию поступившему на вход A_{Map}^{Tm}. При этом если $p^* \in P\_L$, то put(p^*, s^*), иначе, если $p^* \in P$, то $dm_j = p^* s^*$, где p^* определяется из $SHD = P^j Dm_j$, put(dm_j) и $S^* \in BHp$.

2. Если $A_{Map}^{Tm} = 0$, то

2.1) если переменная специального вида dm_k определяет реакцию на шаге s^*, т.е. $dm_k \in rpl(s^*)$, то p определяется из $SHD = P^k Dm_k$ и выбирается шаг следующий за указанным в dm_k воздействии, которое соответствует $rpl(s^*)$ (если таких шагов несколько, то выбор случаен);

2.2) если back $\in rpl(s^*)$, то p = get(back) и выбирается шаг, который следует за указанным в back и воздействие, которое соответствует $rpl(s^*)$ (если таких шагов несколько, то выбор случаен);

2.3) если p $\in rpl(s^*)$, то берем очередной шаг процесса P, если их несколько, то тот, у которого воздействие соответствует реакции (если таких шагов несколько, то выбор случаен);

2.4) если $rpl(s^*)$ содержит обращение к атомарному процессу, то сообщение в $rpl(s^*)$ преобразуется согласно определению данного атомарного

процесса и следующим будет выбран шаг, который следует за s^* в поведении этого процесса, из которого данный s^*;

2.5) если в выбранном процессе нет шага с воздействием, соответствующим $rpl(s^*)$, то работа данного OBS блокируется.

Текущим значением p^* становится s^*.

Рассмотрим свойства наблюдателя на предмет его корректности. Под корректностью наблюдателя мы будем понимать то, что последовательность цепочек шагов, которую порождает OBS, является историей программы, т.е. удовлетворяет отношению R > (P) = R~(P) ∪ R □ (P).

Пусть $S^*(P)$ – множество цепочек в алфавите S(P) и w ∈ $S^*(P)$. Обозначим $[w]_{Pi}$, где $p_i \in P$, последовательность только тех шагов из w, которые принадлежат $S(p_i)$, и в том порядке, в котором они расположены в w, т.е. $[w]_{Pi} \in S^*(p_i)$. Назовем $[w]_{Pi}$ – проекцией w на p_i.

Для обоснования корректности наблюдателя надо показать то, что при $\forall i : p_i \in P, \forall j : OBS_j$ он порождает:

1) цепочку шагов, в которых не нарушены ни R □ (P), (причинно-следственные связи на BHpi), ни R ~ (P) (причинно-следственные связи между процессами), т.е. w удовлетворяет R ~ (P) и $\forall p_i : \exists Hp_i : [w]Pi = Hpi$, где w – цепочка шагов, получаемая при интерпретации. Таким образом, это будет означать, что OBSi корректно воспроизводит реконфигурацию основанную на совместном использовании функциональных элементов для решения ряда подзадач: P1~P2.

2) если процессы прикладных вычислений P распределены между двумя и более OBS, то порождаемые совокупности цепочек $\{w_k\}$ обладают тем свойством, что $p_i : p_i \in P_i, \exists w_k : [w_k]_{P_i} = Hp_i \,\&\, \{w_k\}$ удовлетворяют R ~ (P), т.е. OBS корректно воспроизводит реконфигурацию в форме полной реконфигурации P1 || P2.

Для корректного наблюдателя характерно то, что $\forall i : OBS$ не нарушает отношения R>(P), а это будет тогда, когда для любых SHD и $B_{DE_i}^{LE_i}$ множество

OBS корректно порождает историю вычислительного процесса. Таким образом, введенные нами понятия корректны при корректности прикладных вычислений.

Выводы к разделу 3

В результате проведенных исследований разработана формализованная модель построения и функционирования реконфигурируемых вычислительных устройств. В строго математическом смысле в качестве модели предложена структура с исчислением, правилами вывода. Разработанная модель описывает все характерные особенности функционирования РВУ. В отличии от модели описанной в[4], предложенный нами временный реконфигурируемый исполнитель позволяет описать децентрализацию управления, конфликты на уровне физической среды и задать оба типа реконфигурации. Предложенная форма описания поведения вычислительного процесса отражает независимость вычислений от функционального элемента и времени. Исследована взаимная согласованность и корректность всех введенных в модели понятий. Все это в целом, является основой для создания новой методики проектирования РВУ с более глубокой степенью оптимизации проектных решений.

Таким образом, разработанная формализованная модель построения и функционирования РВУ позволяет описать алгоритмические свойства и поведение реконфигурируемых вычислительных процессов и устройств. Предложенная модель может быть основой для анализа и синтеза реконфигурируемых устройств.

4. ИССЛЕДОВАНИЕ СПОСОБА ПРОЕКТИРОВАНИЯ ЗАЩИТЫ УСТРОЙСТВ НА ПЛИС

Одним методов реализации уникальных идентификаторов FPGA (в том числе и секретных ключей) является применение физически неклонируемых функций. Отличительной особенностью построения цифровых идентификаторов ПЛИС заключается в сравнении числа цифровых импульсов, вырабатываемых двумя функционально идентичными кольцевыми генераторами.

Функциональная схема кольцевого генератора представляет собой нечетное число последовательно соединенных инверторов, при этом выход последнего инвертора, являющийся выходом самого генератора, соединен с входом первого, образуя цепь обратной связи (кольцо). Для определения конкретного начального состояния генератора и обеспечения его управляемости можно добавить в цепь обратной связи двухвходовой логический элемент И, первый вход которого является входом разрешения функционирования генератора, а второй вход участвует в формировании кольца. На рисунок 4.1 представлена обобщенная функциональная модель цифрового кольцевого генератора импульсов.

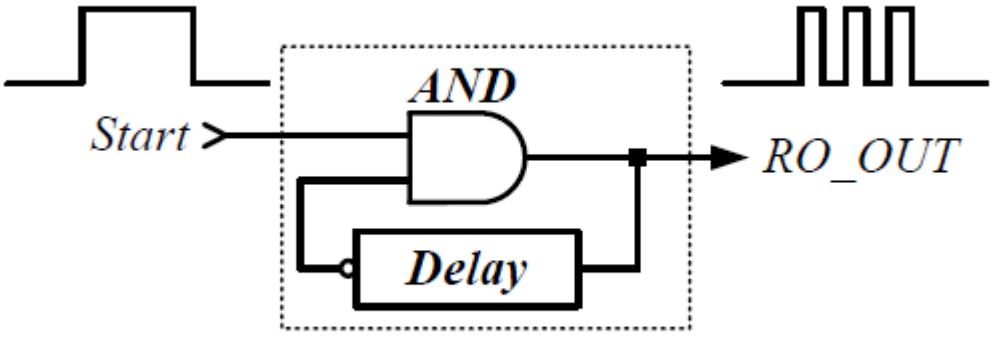


Рис. 4.1 Обобщенная функциональная модель цифрового кольцевого генератора импульсов

Представленный генератор имеет входную линию сигнала разрешения функционирования Start и выходную линию цифровых импульсов RO_OUT. При нулевом значении сигнала Start генератор находится в состоянии ожидания, при этом сигнал на выходной линии принимает также нулевое значение. Функционирование генератора начинается при переключении уровня сигнала Start из значения 0 в значение 1, и продолжается весь период удержания в значении 1. Количество вырабатываемых импульсов на выходе RO_OUT находится в прямой зависимости от продолжительности удержания сигнала на входе Start в значении 1 и от задержки распространения выходного сигнала по цепи обратной связи, включающей логический вентиль AND и элемент задержки Delay (множество инверторов). Обозначим временной интервал удержания сигнала разрешения как Ds, а задержку распространения сигнала в цепи обратной связи генератора как Dro = Dand + Ddelay.

Предположим, что счет импульсов, вырабатываемых генератором, производится цифровым двоичным счетчиком, стробируемым фронтом выходного сигнала RO_OUT. Если Dro > Ds, генератор будет вырабатывать один единственный импульс продолжительностью Ds. В этом случае регистрируемое значение на счетчике будет равно Nr = 1. При условии, что 2Dro < Ds, счетчик будет регистрировать значение Nr > 1. Выразим значение Ds через временные параметры генератора:

$$Ds = Dand + 2Dro(Nr - 1) + \varphi - Dand = 2Dro(Nr - 1) + \varphi$$

где значение φ удовлетворяет следующим неравенствам: $Dand < \varphi < (2Dro - Dand)$.

Будем рассматривать значение Ds; как продолжительность временного окна измерения числа зарегистрированных импульсов Nr.

Для двух сравниваемых генераторов, реализованных на различных ПЛИС, временное окно измерений должно быть идентичным и легко масштабируемым. Предположим, что сигнал Stert вырабатывается для кольцевого генератора на основе внешнего стабильного и технологически

независимого источника синхронизации, которым, например, может служить кварцевый осциллятор.

Условимся, что осциллятор генерирует сигнал синхронизации в форме меандра с частой Fclkg Тогда минимальное значение Ds времени удержания сигнала Start: можно принять равным 1/ Fclkg.

В этом случае для увеличения значения Ds можно использовать аппаратные делители частоты Fclkg, среди которых наиболее предпочтительными выглядят делители с коэффициентом деления $k = 2^i$, реализованные на двоичных счетчиках. При этом временное окно измерения числа импульсов может быть линейно масштабируемым относительно значения Ds.

Так пусть, для увеличения временного окна измерения в четыре раза (k = 4) можно использовать 2-разрядный двоичный счетчик, на вход синхронизации которого поступает сигал с частотой Fclkg = 1/Ds. а на выходе старшего разряда счетчика формируется сигнал Start продолжительностью 4Ds.

Предположим, что измерение числа импульсов производится во временном интервале, кратном значению Ds. В этом случае Ds через временные параметры генератора будет выглядеть следующим образом:

$$kDs = 2Dro(Nr(k) – 1) + \varphi$$

где k есть натуральное число, определяющее коэффициент масштабирования Ds, Nr(k) — число зарегистрированных импульсов, φ(k) — временной интервал, измеряемый от фронта последнего импульса и до момента окончания окна измерения. Покажем, что значение A- непосредственно влияет не только на число регистрируемых импульсов кольцевых генераторов, но и на достоверность идентификации микросхем ПЛИС.

Выразим число регистрируемых импульсов в окне измерения продолжительностью kDs:

$$Nr(k) = \left| \frac{|kDs - \varphi(k)|}{|2\,Dro|} \right| + 1$$

Предположим, что два генератора цифровых импульсов, реализованных на различных ПЛИС, имеют незначительные отличия, которые можно выразить увеличением задержки распространения сигнала по цепи обратной связи D*ro на некоторую малую величину ξ < , D*ro + φ. Тогда, число регистрируемых импульсов второго генератора выражается как

$$Nr^*(k) = \left| \frac{|kDs - \varphi^*(k)|}{|2\,Dro + \xi|} \right| + 1$$

При этом абсолютное значение разности чисел, регистрируемых импульсов двух генераторов оценивается выражением

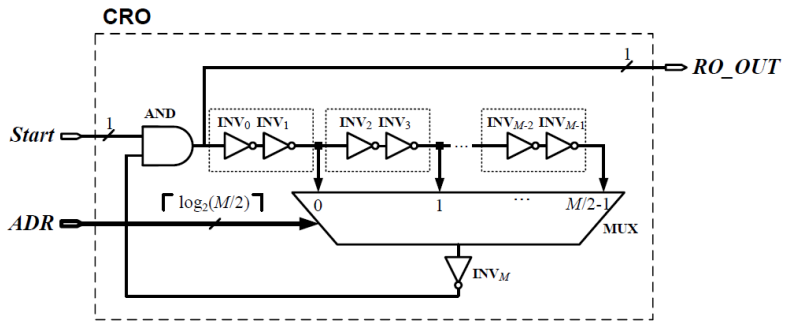
$$\Delta Nr(k) = |Nr(k) - Nr^*(k)| = |\varphi^*(k) - \varphi(k)| / |2Dro|$$

при условии, что ξ < Dro.

Разницу значений ^*(/ - 1). С учетом, что φ*(k) - φ(k) можно представить в виде 2ξ(Nr(k) – 1)

Очевидно, что при малых значениях k разница будет принимать нулевые значения. Установим экспериментальным путем, при каких значениях k она будет принимать стабильные ненулевые значения.

Для проведения эксперимента возмем идентичные цифровые системы Digilant Nexys2 [35], в состав которых входят ПЛИС SPARTAN [31], изготовленные по 90 нм CMOS технологическому процессу. Для возможности реализации генераторов импульсов с различными значениями Ds возмем спроектированную функциональную цифровую модель, структурная схема которой приведена на рисунке.

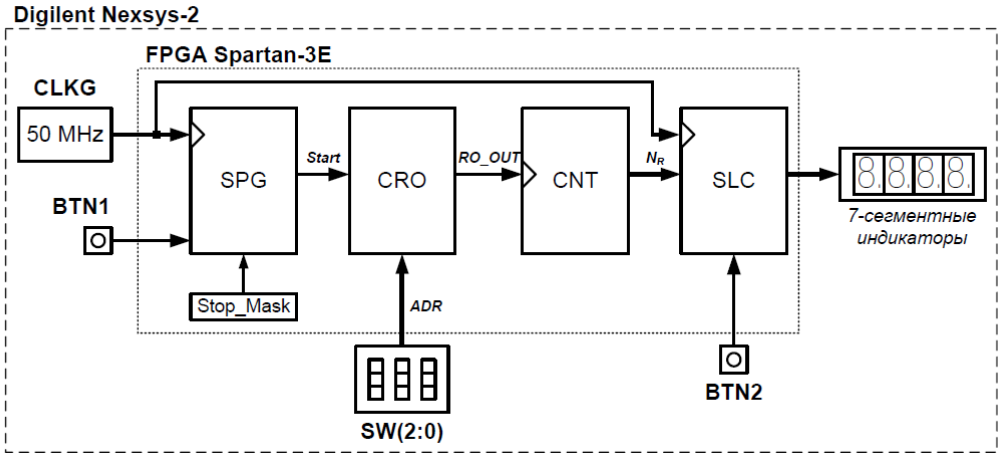


Цепь обратной связи генератора содержит M инверторов, которые попарно сгруппированы. Выход каждой пары подключен к соответствующему входу мультиплексора MUX. Коммутация M/2 входов мультиплексора с его выходным портом определяется двоичным значением, подаваемым на $[1og_2(M/2)]$ -разрядную адресную шину ADR. Выход мультиплексора соединен с инверсным входом логического вентиля AND, выход которого является выходным портом генератора RO_OUT. Таким образом, представленная схема реализует M/2 кольцевых генераторов с различными фиксированными значениями Dro.

Рассмотрим схему конфигурируемого генератора импульсов в качестве схемы, реализующую физически неклонируемую функцию, аргументами которой являются продолжительность импульса на входе Start и значение на входной шине ADR. Значение физически неклонируемой функции есть число зарегистрированных импульсов на выходном порте RO_OUT:

PUFcro = (kDs,ADR) = Nr(k)

Для аппаратной реализации представленной структуры конфигурируемого генератора импульсов было спроектировано цифровое устройство, позволяющее устанавливать различные фиксированные значения kDs и ADR, и с возможностью регистрации вырабатываемых значений Nr(k). Устройство включает в себя следующие основные блоки: генератор стартового импульса SPG, конфигурируемый генератор цифровых импульсов CRO, счетчик импульсов CNT, контроллер светодиодных индикаторов SLC. Устройство, реализуемое на ПЛИС FPGA XC3s500e-5FG320, управляется генератором системных импульсов CLKC (Fclkc = 50 MHx), аппаратными кнопками BTN1 (источник асинхронного сигнала инициализации) и BTN2 (управление режимом отображения результата), тремя переключателями SW(2:0), входящими в состав цифровой системы Digilent Nexys 2. Устройство изображено на рисунке.



Приведенная структура конфигурируемого генератора была спроектирована и описана на языке VHDL с учетом конструктивных особенностей системы Digilent Nexys 2 при помощи САПР ALDEC. По результатам синтеза VHDL-описания аппаратура генератора импульсов занимает 97 Slise-блоков кристалла Xilinx SPARTAN-3E, что составляет около 2 % от всех ресурсов данной ПЛИС (46 Slise-блоков при этом было использовано для реализации контроллера семисегментных индикаторов SLC).

Для проведения эксперимента были взяты две идентичные системы В и В* (Digiland Nexe-2 с ПЛИС FPGA XC3s500e-5FG320), для которых были определены следующие ограничения.

1. Питание обоих систем (+3,3 В) осуществлялось от одного источника питания системного блока ПЭВМ класса Pentium посредством интерфейсных кабелей 115В (+5,0 В) при помощи идентичных регуляторов напряжения LTC1765, входящих в состав систем В и В*.

2. Обе ПЛИС были сконфигурированы одной битовой последовательностью, сгенерированной САПР ALDEC на основе общего VHDL-проекта при абсолютно одинаковых параметрах синтеза, размещения, трассировки соединений и конфигурации.

3. Процедура конфигурирования ПЛИС осуществлялась посредством программного обеспечения ALDEC 5.

4. Выработка системной частоты, на основании которой осуществлялось генерирование сигнала Start, производилась идентичными генераторами 50 МГц.

5. Процессы инициализации и функционирования конфигурируемых генераторов импульсов осуществлялись одновременно.

Обобщая вышесказанное, можно сделать предположение о равных внешних условиях проведения эксперимента для двух функционально идентичных цифровых систем. Эксперимент проводился в несколько этапов. На первом этапе для выбранных значений $k = 2^i$ (i = 0... 10) осуществлялся мониторинг числа сгенерированных импульсов Nr(k) двумя конфигурируемыми генераторами. На рис. 4.1 представлены значения разницы числа зарегистрированных импульсов в зависимости от параметра k и всех возможных восьми значений ADR.

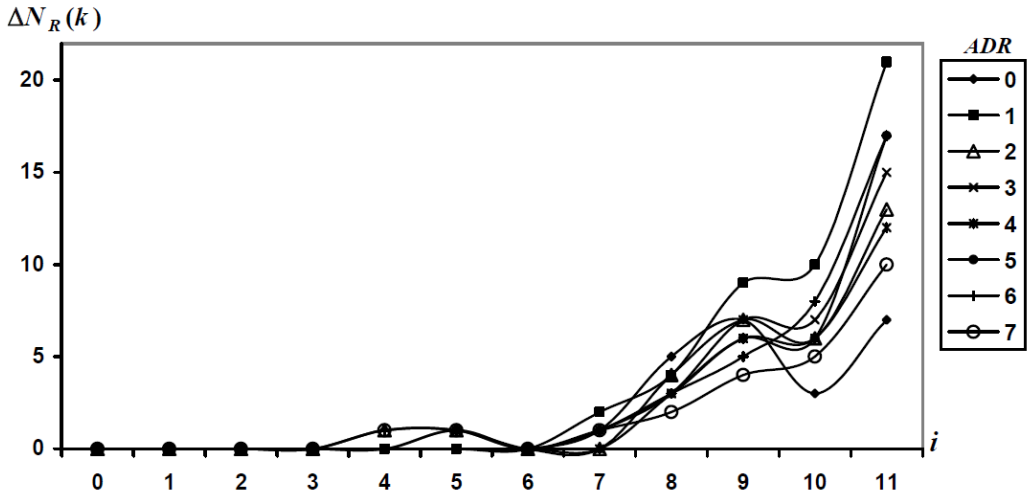


Рис. 4.1 Число импульсов в зависимости от ADR.

Как видно из приведенного графика, первое ненулевое значение разница импульсов принимает для k = 4 и ADR = {2.7}. При этом число импульсов для значения ADR = 2 равно Nr(16) = 23 и Nr*(16) = 22.

С увеличением значения k наблюдается линейное увеличение разницы регистрируемых импульсов для всех значений ADR.

В табл. 4.1 приведены экспериментальные данные, полученные для значения k = 210 (kDs = 210 * 20 = 20480 нс).

Таблица 4.1

Параметры	ADR							
	0	1	2	3	4	5	6	7
$N_R(k)$	7FC	671	55D	4BB	40F	396	33D	2F2
$N_R^*(k)$	7F8	666	557	4B3	408	38F	334	2ED
$D_\Delta(k)$	4	11	6	8	7	7	9	5
$H_\Delta(k)$	1	4	2	1	3	3	2	5

Далее, для каждого значения ADR и выбранного kDs = 20480 не были проведены 100 последовательных экспериментов для выявления отклонений в регистрируемых значениях Nr(k) и Nr*(k). Отклонения в регистрируемых значениях возможны в силу наличия технологических вариаций и выработки генераторами импульсов с частотой превосходящей рабочую частоту функционирования ПЛИС. Так, для значений ADR = 0 и k = 210 регистрируется порядка 2040 импульсов, что соответствует частоте 100 МГц, вдвое превышающей частоту функционирования XC3s500e-5PC320.

В качестве эталонного регистрируемого значения числа импульсов были выбраны наиболее встречаемые значения при 100 последовательных экспериментах. Следует отметить, что для всех возможных значений ADR абсолютная величина отклонений не превышала 2.

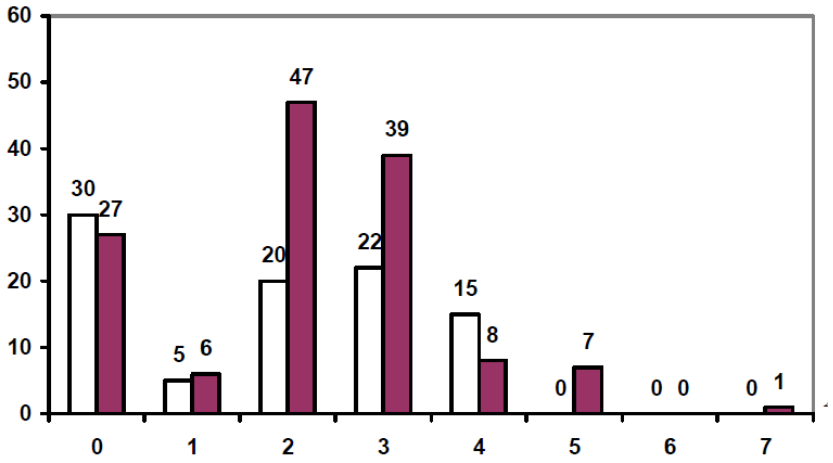


Рис 4.2 Распределение числа отклонений от эталонного значения регистрируемых импульсов.

В свою очередв, значение D(k) определяет число уникальных идентификаторов (число исследуемых микросхем ПЛИС) для выбранного значения k. Например, можно предположить, что для $k = 2^{11}$ около 400 ПЛИС из класса XC3s500e-5P0320 могут быть различимы. Таким образом, увеличение значения k позволяет повышать достоверность идентификации ПЛИС.

Выводы к разделу 4

В данном разделе рассмотрен реализованный способ проектирования защенного устройства на ПЛИС с описанием его работы, приведены и проанализированы возможные атаки на данный метод.

Проанализированы эксперементальные данные для Digiland Nexe-2 с ПЛИС FPGA XC3s500e-5FG320.

Выводы

В дисертации рассмотрены следующие вопросы:

1. Анализ злономаренно созданных аппаратных стедств.

2 Приведен обзор современных методов защиты цифровых проектов и устройств от несанкционированного использования и копирования.

3 Разработан модифицированный способ запутывающих преобразований проектных описаний, внедрения «водяных знаков» и «отпечатков пальцев».

4. Приведен обзор методов аутентификации и идентификации цифровых устройств, реализованных на ПЛИС. Показано, что перспективной технологией, лежащей в основе методов аппаратной аутентификации и идентификации цифровых устройств, является аппаратная реализация физически неклонируемых функций.

5 Исследован разработаный способ использования физически неклонируемой функции кольцевого генератора для решения задачи идентификации FPGA.

Список использованной литературы

1. Adee S. The Hunt for the Kill Switch /Adee S. / IEEE Spectrum, 2008. N5. P. 34-39.

2. King S.T.. Designing and implementing malicious hardware /King S.T.,Tucek J., Cozzie A., Grier C., Jiang W., Zhou Y. / Proceedings of the 1-st Usenix Workshop on Large-Scale

3 Exploits and Emergent Threats, LEET'08. USENIX Association Berkeley, CA, USA, 2008. 8 p.

3. Huffmire T. Handbook of FPGA Design Security /Huffmire T., Irvine C., Nguyen T.D., Levin T., Kastner R., Sherwood T/. –Springer, 2010. – 177 p.

4. B.Badrignans Security Trends for FPGAs. From Secured to Secure Reconfigurable Systems /Editors: B.Badrignans, G.Gogniat, J.L.Danger, L.Torres, V.Fischer/. –Springer, 2011. – 196 p.

5. Lieberman J.I. National security aspects of the global migration of the US semiconductor industry /Lieberman J.I. / White Paper, June 2003. Congressional Record: June 5, 2003 (Senate) – P. S7468-S7471.

6. Obama B.H. Remarks by the President on Securing Our Nation's Cyber Infrastructure (Washington DC, May 29, 2009), режим доступу: http://www.whitehouse.gov/ the_press_office/Remarks-by-the-President-on-Securing-Our-Nations-Cyber-Infrastructure/

7. Secure Semiconductors: Sensible, or Sisyphean? From TIC to IRIS at DARPA. // Defense Industry Daily, 2011. 16 Feb., режим доступу: http://www. defenseindustrydaily.com/Secure-Semiconductors-Sensible-or-Sisiphyean -04928/

8. Hadzic I. FPGA viruses /Hadzic I., Udani S., Smith J., / Proceedings of the 9-th Int. Workshop on Field-Programmable Logic and Applications (FPL'99), Glasgow, UK, August, 1999.

9. Wang X., /Wang X., Tehranipoor M., Plusquellic J/Detecting malicious inclusions in secure hardware: challenges and solutions // Proc. IEEE Workshop on

Hardware Oriented Security and Trust (HOST), Anaheim, CA, June, 2008, 2008. P. 15-19.

10. Trimberger S. Trusted design in FPGAs /Trimberger S. / Proceedings of the 44th Design Automation Conference, San Diego, CA, USA, DAC 2007, June 4–8, 2007, – P. 1.5-1.8.

11. Sarma S.E., /Sarma S.E., Weis S.A., Engels D.W. / Radio-Frequency Identification: Security Risks and Challenges // RSA Laboratories Cryptobytes, Spring 2003. –V.6, N1. – P. 2-9.

12. Котов В.Е. /Котов В.Е. / Введение в теорию схем программ. – Новосибирск-: Наука, 1978. – 256 с.

13. Tiri K., /Tiri K., Verbauwhede I. /Synthesis of Secure FPGA Implementations // Proceedings of International Workshop on Logic and Synthesis, 2004. – P. 224-231.

14. Smith S.C., Di J. Detecting Malicious Logic Through Structural Checking // Proc. IEEE Region 5 Techn. Conf., 20-22 April 2007, 2007. P. 217-222.

15. Виноградов Ю.Н /Виноградов Ю.Н., Иванов Д.Г., Кушниренко Н.В./ Структурный способ иерархической классификации реконфигурируемых вычислительных систем. // Вісник Національного технічного університету України “КПІ” Інформатика, управління та обчислювальна техніка. К.: ТОО “ВЕК+” № 46 . – 2007. С. 133 – 140 .

16 Палагин А.В Реконфигурируемые структуры на ПЛИС /Палагин А.В., Опанасенко В.Н., Сахарин В.Г. / УсиМ. – 2000. - №3 - С. 32 – 39.

17 Хорошевский В.Г. /Хорошевский В.Г. / Инженерный анализ функционирования вычислительных машин и систем. - М.: Радио и связь, 1987.- 256 с.

18 Палагин А.В., Опанасенко В.Н. Реконфигурируемые вычислительные системы./Палагин А.В., Опанасенко В.Н. / – К.: Просвіта, 2006. – 280 с.

19 Чижухин Г.Н., /Чижухин Г.Н., Панферов В.П./ Структурное проектирование вычислительных систем. – Пенза Пенз., политехн. Ин-т, 1985, - 88с.

20 Крон Г. Тензорный анализ сетей: Пер. с англ. / Под ред. Л.Т. Кузина, П.Г. Кузнецова./ – М: Сов. Радио 1987. – 720 с.

21 Plotkin G. An operational semantics for GSP /Plotkin G. / Formal description of programming concepts. N.-H. Press, 1983. p 199 – 223.

22 Котов В.Е. / Котов В.Е/Сети Петри. М.: Наука 1989. – 210 с.