**Содержание**

[Содержание 2](#_Toc194314559)

[Список используемых сокращений 3](#_Toc194314560)

[Введение 4](#_Toc194314561)

[1 Анализ предметной области 5](#_Toc194314562)

[1.1 Обзор аналогов 5](#_Toc194314563)

[1.2 Постановка задачи 7](#_Toc194314564)

[2 моделирование предметной области и разработка функциональных требований 10](#_Toc194314565)

[2.1 Определение физически неклонируемой функции 10](#_Toc194314566)

[2.2 Характеристики ФНФ 14](#_Toc194314567)

[3 Технико-экономическое обоснование разработки программного средства параметрического моделирования физически неклонируемых функций 16](#_Toc194314568)

[3.1 Характеристика программного средства 16](#_Toc194314569)

[3.2 Расчеты затрат на разработку программного средства 16](#_Toc194314570)

[3.3 Оценка эффекта от использования и продажи программного средства 20](#_Toc194314571)

[4 Разработка программного средства 23](#_Toc194314572)

[4.1 Описание проделанной работы 23](#_Toc194314573)

[4.2 Сравнение модели и наборов данных 24](#_Toc194314574)

[Список использованной литературы 30](#_Toc194314575)

[Приложение А 32](#_Toc194314576)

Список используемых сокращений

ФНФ – физически неклонируемая функция.

КО – кольцевой осцилятор.

ПО – программное обеспечение.

ПС – программное средство.

САПР – система автоматического проектирования.

ПЛИС – программируемая логическая интегральная схема.

ASIC – интегральная схема для конкретного применения.

DRAM – динамическая память с произвольным доступом.

SRAM – статическая память с произвольным доступом.

**Введение**

В настоящее время невозможно оценить параметры ФНФ без производства физического образца, а симуляции в САПР дают только поведенческие модели, без характеристики устройства как ФНФ. Производство такого устройства на технологии ASIC стоит больших материальных затрат. А свойства ФНФ при реализации на технологии ПЛИС отличаются от ASIC. В современных САПР нет специализированного функционала, который на основе параметров технологии производства, моделирует и вычисляет характеристики ФНФ.

В данном дипломном проекте рассмотрена попытка создать программное средство для параметрического моделирования физически неклонируемых функций.

1. **Анализ предметной области**
   1. Обзор аналогов

На сегодняшний день существуют множество САПР для проектирования цифровых систем на программируемых логических интегральных схемах (ПЛИС). Каждая из них имеет свои особенности, преимущества и недостатки, что делает их подходящими для различных приложений и предпочтений разработчиков. Для сравнения выделим следующие САПР: Intel Quartus Prime и Xilinx Vivado.

* + 1. Intel Quartus Prime

Intel Quartus Prime представляет собой мощный инструмент для разработки, моделирования и реализации цифровых систем на базе ПЛИС. Симуляция играет ключевую роль в процессе проектирования, позволяя проверять функциональность и производительность систем до их физической реализации.

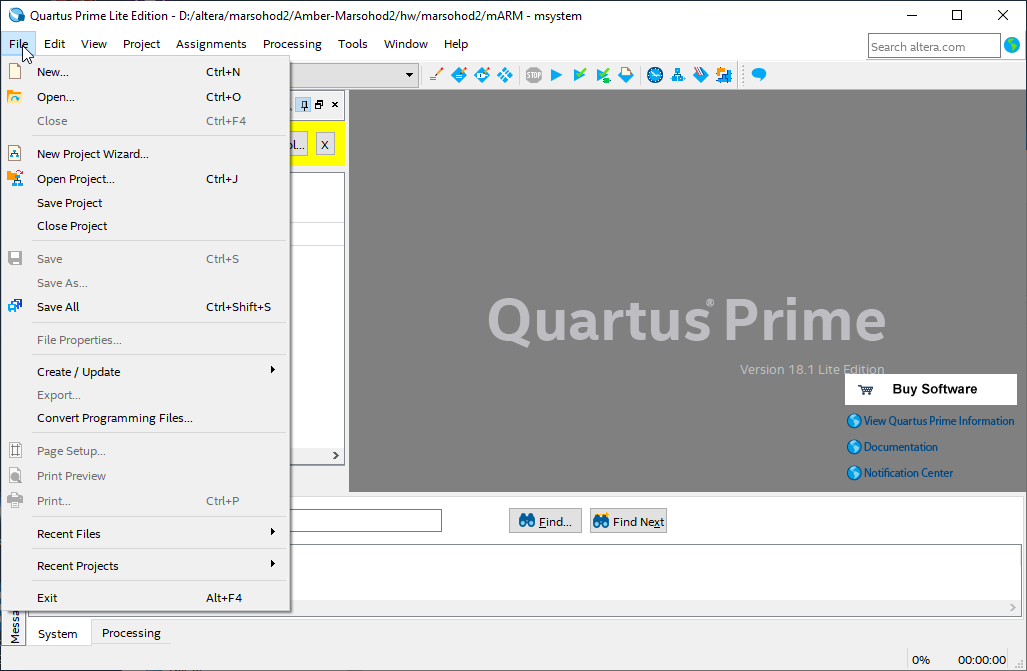


Рисунок 1.1 – интерфейс Intel Quartus Prime

Основным инструментом для симуляции в Quartus Prime является ModelSim, который предлагает возможности функционального и временного моделирования для HDL-кода. Он поддерживает языки описания аппаратуры, такие как VHDL и Verilog, что делает его универсальным для тестирования различных проектируемых функций.

Для проверки работы системы разработчики создают тестовые сценарии, содержащие входные сигналы и ожидаемые выходные результаты. Это позволяет имитировать различные сценарии работы. Quartus Prime облегчает создание и редактирование тестовых сценариев, а также предоставляет возможность генерировать тестовые векторы, что помогает проверить, как система реагирует на различные комбинации входных сигналов.

Инструменты визуализации сигналов в Quartus Prime позволяют разработчикам отслеживать изменения значений сигналов во времени, что способствует выявлению ошибок. В случае неожиданных результатов симуляции Quartus Prime предлагает средства отладки, которые помогают анализировать код и выявлять проблемы, включая функции пошагового выполнения и мониторинга переменных.

Кроме того, Quartus Prime поддерживает временные симуляции, которые учитывают задержки и временные характеристики компонентов. Это критически важно для проверки корректности работы системы в соответствии с заданными временными ограничениями, особенно для физически неклонируемых функций, где задержки могут влиять на безопасность и функциональность. Разработчики могут убедиться, что проект соответствует временным требованиям, указанным в спецификациях.

Также Quartus Prime предоставляет поддержку Universal Verification Methodology (UVM), что позволяет использовать стандартизированные подходы к верификации. UVM упрощает процесс создания сложных тестовых окружений и тестирования. Интеграция с другими системами верификации и инструментами расширяет возможности тестирования проектируемых функций.

* + 1. Xilinx Vivado

Xilinx Vivado предлагает несколько инструментов и подходов для проведения симуляций, включая встроенный Vivado Simulator и поддержку сторонних симуляторов, таких как ModelSim и QuestaSim. Это дает разработчикам возможность использовать более широкие возможности симуляции, доступные в этих инструментах.

Для проверки работы проектируемой функции необходимо создать тестовые сценарии, которые содержат входные сигналы и ожидаемые выходные результаты. Эти тестовые сценарии позволяют имитировать различные сценарии работы системы. Vivado предоставляет возможность генерировать тестовые векторы автоматически или вручную, что включает в себя различные комбинации входных сигналов, помогая проверить, как система справляется с разными условиями.

Vivado позволяет визуализировать сигналы в процессе симуляции, что помогает разработчикам отслеживать изменение значений сигналов во времени и выявлять возможные ошибки. В случае неожиданных результатов симуляции Vivado предлагает мощные средства отладки, позволяя анализировать код и выявлять проблемы. Это может включать пошаговое выполнение кода и мониторинг значений переменных.

Кроме того, Vivado поддерживает временные симуляции, которые учитывают задержки и временные характеристики компонентов. Это важно для проверки корректности работы системы при заданных временных ограничениях, особенно для физически неклонируемых функций, где задержки могут влиять на безопасность и функциональность. Разработчики могут проверять, соответствует ли проект временным требованиям, указанным в спецификациях.

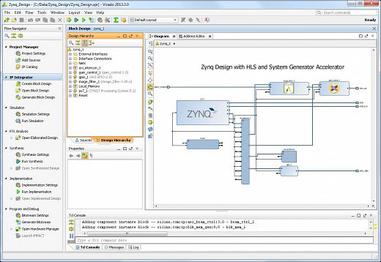


Рисунок 1.2 - Интерфейс Xilinx Vivado

Vivado также поддерживает Universal Verification Methodology (UVM), что позволяет использовать стандартизированные методологии для верификации. UVM упрощает создание сложных тестовых окружений и облегчает процесс тестирования. И нтеграция с другими системами верификации и инструментами расширяет возможности тестирования и проверки проектируемых функций.

* 1. Постановка задачи

В рамках данного дипломного проекта планируется разработать программное средство параметрического моделирования физически неклонируемых функций. Разрабатываемое программное средство будет направлено на моделирование и анализ характеристик физически неклонируемых функций.

* + 1. Назначение разработки

Программное средство должно быть ориентировано не только на выполнение основной функции: моделирование ФНФ, но и иметь приятный пользовательский опыт для занимающихся криптографией аналитиков и инженеров, а так же инженеров по развёртыванию.

Эти группы пользователей привыкли взаимодействовать с ПО разными способами: аналитики любят графические интерфейсы и возможность изменять ход моделирования при помощи собственных скриптов, а инженеры, работающие в сфере микроэлектроники предпочитают лаконичные консольные приложения.

Для выполнения поставленной задачи необходимо:

* определить параметры для моделирования ФНФ,
* составить список интересующих характеристик ФНФ,
* реализовать расчёт характеристик ФНФ,
* разработать архитектуру программного средства,
* разработать классы, реализующие логику системы,
* провести эксперимент на тестовом стенде и сравнить его результаты с выходными данными параметрического моделирования системы.

Целью разработки программного средства является создание удобного и эффективного инструмента для параметрического моделирования ФНФ, позволяющего сократить сроки разработки и дать более точную оценку характеристик ФНФ без реализации устройства. А так же набор разработки ПО для аналитиков, которые могут писать свои ФНФ как плагины.

* + 1. Состав выполняемых функций

Перед разработкой программного средства были определены следующие необходимые для реализации функции:

* Моделирование для технологии ASIC,
* Моделирование для технологии ПЛИС,
* Расчёт метрики единообразия,
* Расчёт метрики уникальности,
* Расчёт метрики стабильности,
* Расчёт метрики надёжности,
* Вычисление соотношения ,
* Построение гистограмм распределения значений метрик и сохранение их в файл;
  + 1. Входные данные

Для корректной работы программного средства необходимо определить перечень входных и выходных данных, с которыми оно будет оперировать. Входные данные представляют собой информацию, вводимую пользователями в систему, а выходные – данные, получаемые в результате обработки входных данных.

Входными данными разрабатываемого в рамках дипломного проекта программного средства являются:

* Количество кольцевых осциляторов в ФНФ
* Мат ожидание распределения периодов кольцевого осцилятора
* Ширина колокола распределения периодов кольцевого осцилятора
* Значение дивиации при сравнении сигналов
* Количество выборок для построения гистограмм распределения
  + 1. Выходные данные

Выходными данными программного средства являются:

* Гистограмма метрики единообразия и основные параметры распределения,
* Гистограмма метрики уникальности и основные параметры распределения
* Гистограмма метрики стабильности и основные параметры распределения
* Гистограмма метрики надёжности и основные параметры распределения
  + 1. Обоснование выбора инструментов разработки

Для реализации программного средства параметрического моделирования физически неклонируемых функций был выбран язык программирования python. Данный выбор обусловлен кросс-платформенностью языка программирования, обширным набором библиотек для работы с анализом данных [1] и их визуализации [2]. Так же python является основным исполнительным ядром в платформе для анализа данных Jupiter Notebook, что позволит аналитикам с лёгкостью использовать пакеты в своих исследованиях. Так же интерпретатор python предустановлен в подавляющем большинстве Lunix дистрибутивов, что позволяет запускать скрипты без дополнительных действий со стороны пользователя.

* + 1. Требования к составу и параметрам технически1х и программных средств

Программное средство будет функционировать локального. Для работы пользователей не требуется доступ к сети интернет. Будет достаточно наличия интерпретатора python в системе, для запуска из среды Jupiter Notebook потребуется соответствующий pip пакет и веб-браузер. Программное средство может быть запущено на операционных системах Windows, macOS или Linux.

1. **моделирование предметной области и разработка функциональных требований**
   1. Определение физически неклонируемой функции
      1. История создания ФНФ

Одной из основополагающих работ в области ФНФ является представленная в 2000 г. работа, использующая невоспроизводимость в кремниевых устройствах для идентификации систем на кристале [3]. Хотя авторы не называли ее ФНФ, цель их работы была очень похожа на цель ФНФ. Примерно в то же время (в 2001 году) Паппу и др. представили концепцию физической односторонней функции, которая привела к идее ФНФ [4]. После этого были предложены различные типы ФНФ. В таблице 1.1 представлен неполный список различных ФНФ по годам их первого упоминания, начиная с 2000 года.

* 1. ФНФ и года их первого упоминания

|  |  |
| --- | --- |
| 2000 | Идентификация систем на кристале [3] |
| 2001 | Физически однонаправленные функции [4] |
| 2002 | Физически случайные функции [5] |
| 2007 | ФНФ типа кольцевой осцилятор [6], SRAM ФНФ [7] |
| 2010 | Глитч ФНФ[8] |

Хотя этот список не является исчерпывающим, он показывает нам картину того, как исследователи пытались построить различные ФНФ. За более подробной информацией можно обратиться к работе, в которой представлено всестороннее описание предложенных на сегодняшний день ФНФ [9].

* + 1. Определение ФНФ

ФНФ является структурой, позволяющей отображать множество запросов *CH* во множество ответов *R* уникальным и невоспроизводимым образом *CH → R*. В цифровых устройствах преобладает использование ФНФ, базирующихся на уникальности и невоспроизводимости задержек распространения сигналов через их пути. Далее будут рассмотрены несколько ФНФ базирующихся на этом эффекте.

* + 1. ФНФ типа кольцевой осцилятор

КО – цифровой генератор, состоящий из нечётного числа инвенторов соединённых последовательно и замкнутых петлёй обратной связи, который колеблется на определённой частоте. Частота колебаний КО определяется особенностью реализации (количество инвенторов или других элементов задержки, маршрутов на кристалле), случайными технологическими отклонениями и условиями эксплуатации, которые оказывают уникальное влияние на каждое устройство.

Традиционная схема ФНФ типа КО предствалена на рисунке 1.

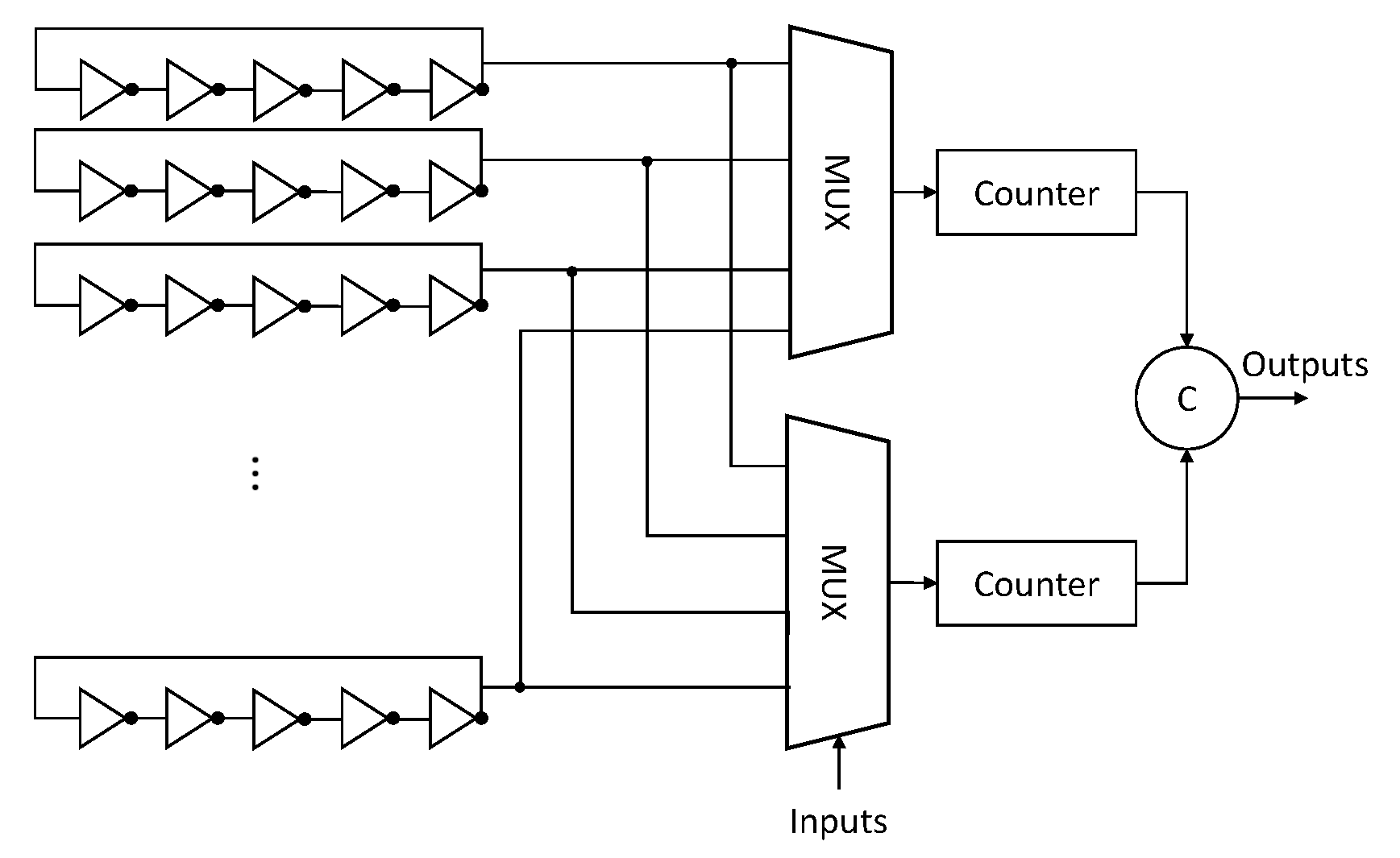


Рисунок 1 – Физически неклонируемая функция типа кольцевой осцилятор

Она измеряет частоты пары КО с помощью двух счётчиков и сравнивает эти две частоты, чтобы сформировать бит ответа на основе следующего правила:

N -битная сигнатура может быть создана путём сравнения нескольких КО. Обычно эти КО реализуются в двумерной матрице.

* + 1. ФНФ типа арбитр

На рисунке 2 показана схема ФНФ на основе мультиплексоров и арбитра. Схема имеет многобитный вход и 1-битный выход на основе относительной задержки разницы распространения сигнала между двумя путями. [10]

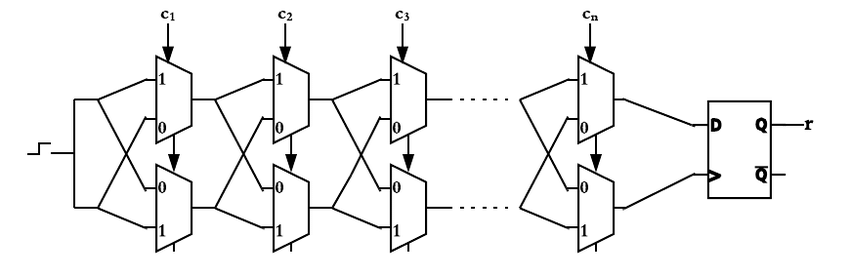


Рисунок 2 – Физически неклонируемая функция типа арбитр

Входные биты управляют мультиплексорами и определяют конфигурацию путей по которым проходит сигнал. Мультиплексоры пропускают два сигнала задержки с левой стороны, если входной управляющий бит равен нулю. В противном случае верхний и нижний сигнал меняется местами. Таким образом, схема выбирает пару путей для каждого бита закпроса. Сигнал проходит по двум трактам путей и арбитр (защелка) в конце решает какой сигнал быстрее. На выходе получается единица, если сигнал на вход данных защелки быстрее, и ноль в противном случае.

* + 1. ФНФ типа SRAM

ФНФ типа SRAM основывается на эффекте инициализации ячейки памяти случайным значением. Чтобы лучше понять этот эффект рассмотрим структуру типичной шеститранзисторной КМОП-ячейки SRAM [11], как показано на рис. 3. Такая ячейка состоит из двух перекрестно соединенных инверторов (транзисторы PL, PR, NL и NR) и двух транзисторов доступа (AXL и AXR), подключаемых к битовым линиям данных (BLC и BL) на основе сигнала линии слова (WL). Каждый инвертор состоит, в свою очередь, из транзистора с p-переходом (PL, PR) и транзистора с n-переходом (NL, NR) [12].

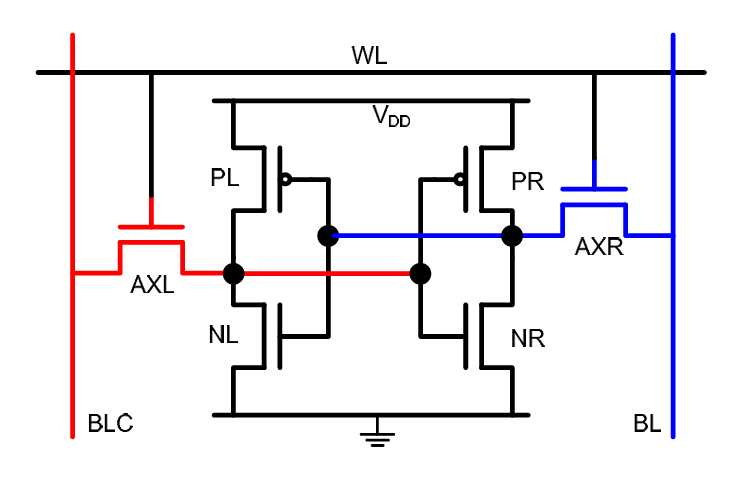


Рисунок 3 – структура SRAM ячейки

Для каждой ячейки SRAM-памяти существует свойство, известное как статический шумовой запас. Оно соответствует амплитуде напряжения шума, который может стереть и перезаписать состояние ячейки, потеряв хранящуюся в ней информацию. Амплитуда шума, вызывающего это изменение, зависит от производственных вариаций каждой ячейки. На основе этого можно сформировать ФНФ. К массиву ячеек прикладывается вызов *CH* в виде постепенно увеличивающегося шума. Это добавляет к напряжению шум, чтобы эффективно снизить статический шумовой запас на все большую величину. При определенном приложенном напряжении порог статического шумового запаса для определенной ячейки достигается, и данные, хранящиеся в SRAM, сбрасываются и теряются. Прикладывая это напряжение к большому массиву элементов SRAM и определяя, какие элементы выходят из строя первыми, можно построить стабильную характеристику массива, которая будет подобна ФНФ.

В данном случае запросом (*CH*) ФНФ будет напряжение, приложенное к массиву ячеек SRAM, а ответом (*R*) - то, в каких ячейках этого массива в результате произошел индуцированный сбой битов.

* + 1. ФНФ типа DRAM

Базовая ячейка DRAM состоит из одного накопительного конденсатора, отделенного от остальной части системы одним транзистором, как на рис. 4.

Когда ячейка DRAM инициализируется при подаче питания, она не обязательно инициализируется в состояние 0 (разряжена), как можно было бы ожидать. Это происходит потому, что при запуске конденсатор предварительно заряжаются до половины напряжения накопителя. Такая предварительная зарядка делает теоретически одинаково вероятным, что конденсатор перейдет в состояние 1 или 0.

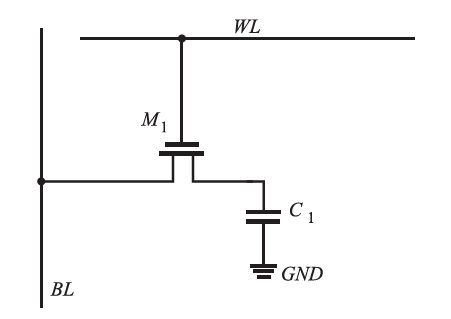


Рисунок 4 – ячейка DRAM

Фактически инициализируемое значение ячейки, определяется неконтролируемыми различиями в производстве каждой ячейки. На этом эффекте и основывается ФНФ типа DRAM. В этом случае запросом *CH* будет номер или позиция ячейки DRAM, а ответом *R* – состояние ячейки. Слабым местом ФНФ типа DRAM является необходимость включения и выключения для переинициализации.

* + 1. Сильные и слабые ФНФ

Слабые ФНФ можно рассматривать как своего рода память, которая случайным образом инициализируется при включении питания. Запрос (*CH*) можно рассматривать как адрес в памяти, а ответ (*R*) - как случайное значение, хранящееся по этому адресу. Таким образом, количество уникальных пар «вызов-ответ» (*CH → R*) линейно зависит от размеров памяти. Слабость заключается в том, что количество *CH → R* невелико и может быть быстро исчерпано.

Сильные ФНФ - это системы, генерирующие ответы на основе своей внутренней структуры. Количество уникальных пар *CH → R* в них растет быстрее, чем линейно, с увеличением количества случайных элементов из-за взаимодействий между ними. Преимущество заключается в том, что таким образом можно сделать пространство *CH → R* достаточно большим, чтобы сделать его исчерпание практически невозможным, а совпадение двух случайно выбранных элементов пространства достаточно маловероятными. Еще одним преимуществом является то, что случайность может храниться не только в самих элементах, но и в их взаимодействиях, которые иногда невозможно прочитать напрямую. Слабость заключается в том, что одни и те же элементы и их взаимодействия повторно используются для разных задач, что открывает возможность получить некоторую информацию об элементах и их связях и использовать ее для предсказания реакции системы на ненаблюдаемые задачи.

В научной литературе нет чёткого количества пар *CH → R*, когда ФНФ считается сильной, а не слабой. Но как правило во всех работах сильными называли ФНФ с количеством пар больше чем .

* 1. Характеристики ФНФ

В данном разделе будут рассмотрены характеристики ФНФ и способы их расчёта, которые обязательны к реализации в программном средстве.

* + 1. Единообразие

Двоичная ФНФ является единообразной, если n-битовый ответ содержит равное количество 0 и 1. Однородность можно рассчитать следующим образом [13, 14]:

|  |  |
| --- | --- |
| , | (2.1) |

где – представляет собой *l*-й бит *n*-битового ответа ПЛИС *i*;

* + 1. Уникальность

Уникальность - это метрика, отражающая способность ФНФ различать два устройства. По сути, она измеряет, насколько различаются ответы устройств на один и тот же запрос. Математически уникальность можно описать следующим образом [13, 14]:

|  |  |
| --- | --- |
| , | (2.2) |

где *k* – количество ПЛИС;

*HD* – расстояние по Хэммингу между ответами ПЛИС под индексом *j* и *i*;

Идеальное значение этой метрики является 0.5*.*

* + 1. Надёжность

Надежность отклика с течением времени - важнейший аспект ФНФ. Генерирование одного и того же отклика в различных условиях эксплуатации и окружающей среды, а также после длительной эксплуатации в полевых условиях (старение) имеет важное значение для ПУФ. Надежность обычно оценивается количественно, как описано ниже [13, 14]:

|  |  |
| --- | --- |
| , | (2.3) |

где *x* – количество образцов *n*-битного ответа;

– расстояние Хэмминга между *y* образцом и эталонным ответом;

Идеальное значение метрики надёжности ответов равняется единице.

* + 1. Стабильность

Метрика стабильности сильно зависит от надёжности. И рассчитывается по следующему алгоритму:

|  |  |
| --- | --- |
| , | (2.4) |

где n – количество битовых векторов с ответами от ПЛИС;

– вектор ответа от ПЛИС;

После каждый элемент вектора *res* модифифицируется по правилу:

где – элемент вектора под индексом *i*;

– такой же как в формуле 2.4;

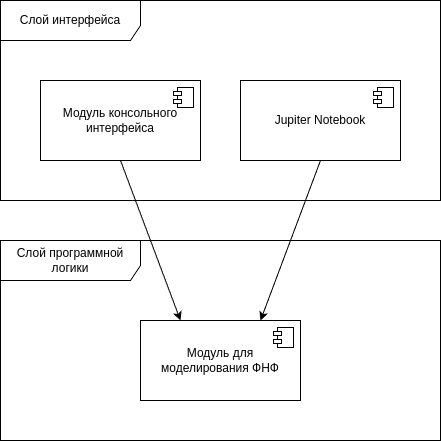
1. Проектирование программного средства
   1. Разработка архитектуры программного средства

Разрабатываемое программное средство представляет собой модуль на языке Python, который предоставляет основной функционал в виде функций и классов, а также консольную утилиту для взаимодействия с пользователем через командную строку. Такой подход обеспечивает универсальность использования: модуль может быть интегрирован в другие проекты или использоваться для интерактивного анализа данных в среде Jupyter Notebook.

Архитектура программного средства основана на принципах модульности и повторного использования кода. Основная логика реализуется в виде отдельного Python-модуля, который содержит все необходимые функции и классы. Консольная утилита реализует пользовательский интерфейс командной строки и использует функционал основного модуля.

* + 1. Взаимодействие компонентов

Консольная утилита импортирует функции и классы из core.py и предоставляет пользователю удобный способ вызова этих функций через командную строку. Использование в Jupyter Notebook: пользователь может импортировать функции из модуля моделирования напрямую и использовать их в интерактивных ячейках.



aaaaaaaaaaaaaaaa

1. **Технико-экономическое обоснование разработки программного средства параметрического моделирования физически неклонируемых функций**
   1. Характеристика программного средства

Программное средство для параметрического моделирования физически неклонируемых функций – это специализированный набор исполняемых файлов, которые позволяют смоделировать ФНФ имея характеристики технологии производства.

Основная задача программного средства на основе параметров производственной технологии промоделировать и проанализировать параметры ФНФ и графически показать результаты. Пользователи с могут анализировать характеристики ФНФ без изготовления прототипа, что ускоряет разработку ФНФ и значительно снижает затраты.

Для потребителей криптографических чипов, которые используются практически во всех устройствах, это означает их удешевление. Для предприятий внедрение данного программного средства способствует, снижению эксплуатационных расходов и увеличению общей эффективности производственных процессов.

Современные САПР, такие как Xilinx Vivado и Intel Quartus не позволяют промоделировать и предоставить отчёт об характеристиках ФНФ. Разрабатываемое программное средство будет иметь такую возможность, что является несомненным преимуществом.

* 1. Расчеты затрат на разработку программного средства

Расчет затрат на разработку ПО производится в разрезе следующих статей затрат:

* затраты на основную заработную плату разработчиков;
* затраты на дополнительную заработную плату разработчиков;
* отчисления на социальные службы;
* прочие затраты (амортизационные отчисления, расходы на электроэнергию, командировочные расходы, арендная плата за офисные помещения и оборудование, расходы на управление и реализацию и т.п.).
  + 1. Затраты на основную заработную плату команды разработчиков делятся исходя из численности, состава команды (категорий исполнителей), размеров месячной заработной платы каждого из участников команды, а также общей трудоемкости разработки ПО.

Расчет основной заработной платы осуществляется по формуле 7.1:

|  |  |  |
| --- | --- | --- |
|  |  | (7.1) |

где n – категории исполнителей, занятых разработкой программного средства;

Зч – часовой оклад исполнителя i-й категории, руб.;

t – трудоемкость работ, выполняемых исполнителем i-й категории, ч.

Согласно постановлению Министерства труда и социальной защиты Республики Беларусь от 15 ноября 2024 г. № 67 «Об установлении расчетной нормы рабочего времени на 2024 год» при полной норме продолжительности рабочего времени на 2025 г. для пятидневной рабочей недели с выходными днями в субботу и воскресенье расчетная норма рабочего времени составит 2007 ч. На основании этих данных среднее количество рабочих ч. в месяце принято равным 167 ч.

Трудоемкость определялась на основе сложности разработки программного средства, объема функций. За основу в том числе брались фактические значения трудоемкости работ при разработке ПО со схожим функционалом в месте прохождения преддипломной практики.

При расчете размер премии принимается 20% – это фактически сложившееся значение в месте прохождения преддипломной практики.

На основании плановых данных был выполнен расчет основной заработной платы команды разработчиков, результаты которого приведены в таблице 7.1.

* 1. Расчет основной заработной платы команды разработчиков

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Наименование должности разработчика | Вид выполняемой работы | Месячная заработная плата, р. | Часовая заработная плата, р. | Трудоёмкость работ, ч. | Сумма, р. |
| Инженер-программист | Разработка программного средства | 2200 | 13,17 | 320 | 4214,40 |
| Специалист по тестированию программного средства | Тестирование программного средства | 1600 | 9,58 | 160 | 1532,80 |
| Итого | | | | | 5747,20 |
| Премия (20%) | | | | | 1149,44 |
| Всего основная заработная плата | | | | | 6896,64 |

* + 1. Расчет затрат на дополнительную заработную плату команды разработчиков включают выплаты, предусмотренные законодательством о труде (оплата трудовых отпусков, льготных ч., времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей), и определяется по формуле 7.2:

|  |  |  |
| --- | --- | --- |
|  |  | (7.2) |

где Нд – норматив дополнительной заработной платы;

Зо – затраты на основную заработную плату, р.

Согласно ст.153,155 Трудового кодекса Республики Беларусь трудовой отпуск предоставляется за работу в течение рабочего года (ежегодно) с сохранением прежней работы и среднего заработка. Продолжительность основного отпуска не может быть менее 24 календарных дней.

Наниматель по месту прохождения практики работникам с ненормированным рабочим днем также устанавливает дополнительный отпуск за ненормированный рабочий день продолжительностью до 7 календарных дней.

Норматив дополнительной заработной платы рассчитываем, как отношение рассчитанной суммы отпускных за 31 календарный день к сумме начисленной заработной платы за 11 месяцев, и принимаем равным 10%.

Тогда размер дополнительной заработной платы исполнителей составит:

* + 1. Отчисления на социальные нужды производятся от фонда заработной платы и включают в себя отчисления в фонд социальной защиты населения – 35%. Определяются в соответствии с действующими законодательными актами по формуле 7.3:

|  |  |  |
| --- | --- | --- |
|  |  | (7.3) |

где Нсоц – норматив отчислений от фонда оплаты труда.

Размер отчислений в фонд социальной защиты населения составит:

* + 1. Расчет затрат на спецоборудование для тестового стенда для сбора эксперементальных данных и их последующего анализа представлены в таблице
  1. Расчет затрат на спецоборудование

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование специальных инструментов, приспособлений, приборов, стендов, устройств и другого специального оборудования | Количество, шт. | Цена за единицу, р. | Сумма, р. |
| 1. Zybo Z7: Zynq-7000 | 4 | 1500 | 6000 |
| 2. Моноблок ASUS E3402WVA-WPC0130 | 2 | 2645,20 | 5290,40 |
| Всего | | | 11290,40 |

* + 1. Включают затраты, как напрямую связанные с разработкой конкретного программного продукта в соответствии с планируемой суммой затрат на эти мероприятия, так и затраты, связанные с функционированием организации-разработчика в целом. Данные затраты включаются в себестоимость разработки ПО в процентах от затрат на основную заработную плату разработчиков по формуле (7.4):

|  |  |  |
| --- | --- | --- |
|  |  | (7.4) |

где Нпз – норматив прочих затрат (100%).

Тогда расходы на прочие затраты составят:

Полная сумма затрат на разработку программного средства находится путём суммирования всех рассчитанных статей затрат. Результаты вычислений представлены в таблице 7.2.

* 1. Затраты на разработку программного средства

|  |  |
| --- | --- |
| Наименование статьи затрат | Значение, р. |
| Основная заработная плата команды разработчиков | 6896,64 |
| Дополнительная заработная плата команды разработчиков | 689,66 |
| Отчисления на социальные нужды | 2731,07 |
| Прочие затраты | 6896,64 |
| Затраты на спецоборудование | 11290,40 |
| Общая сумма затрат на разработку | 28504,41 |

Общая сумма затрат на разработку программного средства обработки вибросигнала составит 28504,01 р.

* 1. Оценка эффекта от использования и продажи программного средства
     1. Неэкономический эффект от продажи программного средства

Неэкономический эффект от продажи программного средства параметрического моделирования физически неклонируемых функций представляет собой более точное представление параметров ФНФ, что позволяет снижать связанные с характеристиками риски для разработки включающих ФНФ в свой состав устройств. А так же позволяет значительно ускорить проверку гипотез о ФНФ путём моделирования, а не изготовления и анализа характеристик прототипа.

* + 1. Экономический эффект от продажи программного средства

Экономический эффект организации-разработчика программного обеспечения в данном случае представляет собой прибыль от продажи ПО множеству потребителей. Прибыль от реализации напрямую зависит от объёмов продаж, цены реализации и затрат на разработку данного ПО.

При определении цены реализации требуется учесть, что стоимость зависит от множества факторов и имеет большой разброс. В случае разработанного программного продукта имеем средство, в виде утилиты для инженеров. В странах СНГ не столь развито предоставление компаниям САПР или производственных утилит. Предпочтение отдается созданию под заказ индивидуальных средств. На основе открытых данных о стоимости программного обеспечения подобного класса в США, где средняя стоимость покупки лицензии Xilinx Vivado на год составляет 5000 долл. США для одной рабочей станции. В данную стоимость входит предоставление, поддержка и обновление ПО. Сделаем дисконт в 40% для рынка СНГ и примем цену ПО за 10500 р.

Анализ рынка стран СНГ, которые являются ключевыми в вопросе продажи данного программного средства, показал, что на рынке есть порядка 200 компаний проектировщиков чипов. Предположим, что 20% из этих компаний нуждаются в программном средстве. Исходя из этого имеем около 40 потенциальных покупателей. Можем сделать предположение, что на старте продаж можно рассчитывать на показатель в 10 подписок в год.

Прибыль, полученная разработчиком от реализации ПО на рынке, рассчитывается по формуле 7.5:

|  |  |  |
| --- | --- | --- |
|  |  | (7.5) |

где Ц – стоимость подписки сроком на один год, р;

N – количество компаний, купивших подписку за год;

НДС – налог на добавленную стоимость, р.;

– общая сумма затрат на разработку программного продукта, р.;

– рентабельность продаж подписки, принимаем 40%.

Величину налога на добавленную стоимость с доходов от реализации приложения можно определить по формуле 7.6:

|  |  |  |
| --- | --- | --- |
|  |  | (7.6) |

где – ставка налога на добавленную стоимость, согласно действующему законодательству, составляющая 20%.

Налог на добавленную стоимость, рассчитанный согласно формуле 7.6, составит:

Таким образом, рассчитанная по формуле 7.5 прибыль составит:

Рассчитаем чистую прибыль, используя формулу 7.7:

|  |  |
| --- | --- |
| , | (7.8) |

где – ставка налога на прибыль, равная 20%.

С учетом ранее рассчитанной прибыли получим:

Для оценки эффективности вложения средств в разработку программного продукта необходимо рассчитать уровень рентабельности затрат. Расч ет данного показателя может быть выполнен по формуле 7.8:

|  |  |
| --- | --- |
| . | (7.8) |

С учетом ранее рассчитанных значений затрат на разработку и прибыли организации уровень рентабельности составит:

Поскольку средняя процентная ставка по банковским депозитным вкладам на январь 2025-го г. не превышает 13,76% [15], данный проект программного средства параметрического моделирования ФНФ является экономически эффективным, разработка и последующая продажа программного продукта являются экономически целесообразными.

В результате технико-экономического обоснования разработки программного средства параметрического моделирования ФНФ были получены следующие экономические показатели: рентабельность составляющая 18,23%, что существенно превышает среднюю процентную ставкой по банковским депозитным вкладам на январь 2025-го г. (ставка составляет 13,76%). Рентабельность рассчитана при условии, что стоимость продажи лицензии на год составит 10500 р, а планируемые продажи в год равняются 10. Прогнозируемая сумма затрат на разработку программного продукта составляет 28504,41 р. Чистая прибыль от реализации программного средства была предварительно оценена в сумму Исходя из данных показателей, можно сделать вывод о привлекательности разработанной системы для рынка.

1. Разработка программного средства
   1. Описание проделанной работы

За время практики был разработан python модуль, который послужит основой программмного средства для дипломного проекта. Данный модуль позволил промоделировать ФНФ типа кольцевой осцилятор.учитывая следующие параметры:

* Значение дивиации при сравнении сигналов (PD\_DEVIATION\_SIGMA)
* Количество осциляторов в ФНФ (PD\_COUNT)
* Мат ожидание величины задержки (PD\_CENTER)
* Ширина колокола распределения задержек (PD\_SIGMA)
* Количество сэмплов для гистограмм распределения (NUMBER\_OF\_RESPONSES)

Так же при помощи модуля был проведён анализ предоставленных наборов данных с кластера ПЛИС. Кластер состоял из 4 ПЛИС и на них был произведён сбор данных в следующей конфигурации: На каждой ПЛИС было реализовано по 4 ФНФ типа КО и измерены частоты их внутренних КО.

**Список использованной литературы**

[1] Matplotlib. [Электронный ресурс]. – 2025 – Режим доступа: <https://matplotlib.org/>

[2] pandas - Python Data Analysis Library. [Электронный ресурс]. – 2025 – Режим доступа: <https://pandas.pydata.org/>

[3] K. Lofstrom, W. Daasch, and D. Taylor. Ic identification circuit using device mismatch. In Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International, pages 372–373, 2000. [Электронный ресурс]. – 2025 – Режим доступа: https://ieeexplore.ieee.org/abstract/document/839821

[4] R. S. Pappu, B. Recht, J. Taylor, and N. Gershenfeld. Physical one-way functions. Science, 297:2026–2030, 2002. [Электронный ресурс]. – 2025 – Режим доступа: <https://cba.mit.edu/docs/theses/01.03.pappuphd.powf.pdf>

[5] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Silicon physical random func tions. In Proceedings of the 9th ACM conference on Computer and communications security, CCS ’02, pages 148–160, New York, NY, USA, 2002. ACM. [Электронный ресурс]. – 2025 – Режим доступа: <https://safari.ethz.ch/architecture_seminar/fall2018/lib/exe/fetch.php?media=10.1.1.297.5196.pdf>

[6] G. E. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. In Proceedings of the 44th annual Design Automation Conference, DAC ’07, pages 9–14, New York, NY, USA, 2007. ACM. [Электронный ресурс]. – 2025 – Режим доступа: <https://people.csail.mit.edu/devadas/pubs/puf-dac07.pdf>

[7] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls. Fpga intrinsic pufs and their use for ip protection. In Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems, CHES ’07, pages 63–80, Berlin, Heidelberg, 2007. Springer-Verlag. [Электронный ресурс]. – 2025 – Режим доступа: <https://iacr.org/archive/ches2007/47270063/47270063.pdf>

[8] D. Suzuki and K. Shimizu. The glitch puf: a new delay-puf architecture exploit ing glitch shapes. In Proceedings of the 12th international conference on Crypto graphic hardware and embedded systems, CHES’10, pages 366–382, Berlin, Heidel berg, 2010. Springer-Verlag. [Электронный ресурс]. – 2025 – Режим доступа: <https://www.iacr.org/archive/ches2010/62250356/62250356.pdf>

[9] R. Maes and I. Verbauwhede. Physically unclonable functions: A study on the state of the art andfuture research directions. In Towards Hardware-Intrinsic Security. Springer, 2010. [Электронный ресурс]. – 2025 – Режим доступа: <https://www.researchgate.net/publication/226371108_Physically_Unclonable_Functions_A_Study_on_the_State_of_the_Art_and_Future_Research_Directions>

[10] D. Lim, Extracting Secret Keys from Integrated Circuits, Master’s Thesis, MIT, 2004 [Электронный ресурс]. – 2025 – Режим доступа: <https://www.researchgate.net/publication/3337710_Extracting_secret_keys_from_integrated_circuits>

[11] Torrens, G.; Alheyasat, A.; Alorda, B.; Bota, S.A. SRAM-Based PUF Reliability Prediction Using Cell-Imbalance Characterization in the State Space Diagram. Electronics 2022, 11, 135. [Электронный ресурс]. – 2025 – Режим доступа: <https://doi.org/10.3390/electronics11010135>

[12] Guajardo, J., Kumar, S.S., Schrijen, GJ., Tuyls, P. (2007). FPGA Intrinsic PUFs and Their Use for IP Protection. In: Paillier, P., Verbauwhede, I. (eds) Cryptographic Hardware and Embedded Systems - CHES 2007. CHES 2007. Lecture Notes in Computer Science, vol 4727. Springer, Berlin, Heidelberg. [Электронный ресурс]. – 2025 – Режим доступа: <https://doi.org/10.1007/978-3-540-74735-2_>5

[13] Maiti, A., Gunreddy, V., Schaumont, P. (2013). A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions. In: Athanas, P., Pnevmatikatos, D., Sklavos, N. (eds) Embedded Systems Design with FPGAs. Springer, New York, NY. [Электронный ресурс]. – 2025 – Режим доступа: <https://doi.org/10.1007/978-1-4614-1362-2_11>

[14] Martin H, Peris-Lopez P, Natale GD, Taouil M, Hamdioui S. Enhancing PUF Based Challenge–Response Sets by Exploiting Various Background Noise Configurations. Electronics. 2019; 8(2):145. [Электронный ресурс]. – 2025 – Режим доступа: <https://www.mdpi.com/2079-9292/8/2/145>

[15] Динамика ставок кредитно-депозитного рынка [Электронный ресурс]. – 2025 – Режим доступа: <https://www.nbrb.by/statistics/creditdepositmarketrates>

Приложение А

(обязательно)

Код программы

import numpy as np

import matplotlib.pyplot as plt

from scipy.spatial import distance

from distfit import distfit

def calc\_metrics(pds: list, number\_of\_responses, pd\_deviation\_sigma) -> list:

unif = \_\_analyze\_unif(pds[0], number\_of\_responses, pd\_deviation\_sigma)

stable = \_\_analyze\_stable(pds, number\_of\_responses, pd\_deviation\_sigma)

rel = \_\_analyze\_reliability(pds, number\_of\_responses, pd\_deviation\_sigma)

uniq = \_\_analyze\_uniqueness(pds, number\_of\_responses, pd\_deviation\_sigma)

return [unif, stable, rel, uniq]

def plot\_metrics(metrics: list):

figure, axs = plt.subplots(2,2)

figure.set\_size\_inches(20, 15)

axs = axs.ravel()

dfit = distfit(distr="norm")

dfit.fit\_transform(metrics[0])

dfit.plot(ax=axs[0])

axs[0].set\_title("Uniformity")

dfit.fit\_transform(metrics[1])

dfit.plot(ax=axs[1])

axs[1].set\_title("Stable")

dfit.fit\_transform(metrics[2])

dfit.plot(ax=axs[2])

axs[2].set\_title("Reliability")

dfit.fit\_transform(metrics[3])

dfit.plot(ax=axs[3])

axs[3].set\_title("Uniqueness")

def \_\_analyze\_unif(pd: np.array, number\_of\_responses: int, pd\_deviation\_sigma: float) -> np.array:

responses = []

for \_ in range(number\_of\_responses):

pairs = \_\_select\_pairwise(pd)

bit\_vector = \_\_calc\_bit\_vector(pairs, pd\_deviation\_sigma)

responses.append(bit\_vector)

total = np.zeros(len(responses), dtype=np.float32)

for i in range(len(responses)):

# total[i] = calc\_uniformity(responses[i])

total[i] = \_\_calc\_uniformity\_normalized(responses[i])

return total

def \_\_analyze\_stable(pds: list, number\_of\_responses: int, pd\_deviation\_sigma: float) -> np.array:

res = np.zeros(len(pds), dtype=np.float32)

for i, pd in enumerate(pds):

responses = [ \_\_calc\_bit\_vector(\_\_select\_pairwise(pd), pd\_deviation\_sigma) for \_ in range(number\_of\_responses) ]

res[i] = \_\_calc\_stable(responses)

print(res[i])

return res

def \_\_analyze\_reliability(pds: list, number\_of\_responses: int, pd\_deviatio\_sigma: float) -> np.array:

total = np.zeros(len(pds), dtype=np.float32)

for cur\_iter, pd in enumerate(pds):

pairs = \_\_select\_pairwise(pd)

responses = [

\_\_calc\_bit\_vector(pairs, pd\_deviatio\_sigma) for \_ in range(number\_of\_responses)]

total[cur\_iter] = \_\_calc\_reliability(responses)

return total

def \_\_analyze\_uniqueness(pds: list, number\_of\_responses: int, pd\_deviation\_sigma: float) -> np.array:

total = []

for \_ in range(number\_of\_responses):

fpga\_resps= [ \_\_calc\_bit\_vector(\_\_select\_pairwise(pd), pd\_deviation\_sigma) for pd in pds ]

result = \_\_calc\_uniquness(fpga\_resps)

total.append(result)

return np.array(total)

def \_\_select\_pairwise(input\_array: np.array) -> np.array:

x = np.array(np.meshgrid(input\_array, input\_array)).T.reshape(-1, 2)

y = np.delete(x, np.arange(0, len(input\_array)\*\*2, len(input\_array)+1), axis=0)

return y

def \_\_calc\_bit\_vector(pairs: np.array, pd\_deviation\_sigma: int) -> np.array:

kek = np.hsplit(pairs, 2)

kek0 = kek[0].flatten() \* \_\_get\_deviation(len(kek[0]), pd\_deviation\_sigma)

kek1 = kek[1].flatten() \* \_\_get\_deviation(len(kek[0]), pd\_deviation\_sigma)

result = (kek0 > kek1).astype(int)

return result

def \_\_get\_deviation(size: int, pd\_deviation\_sigma: float) -> np.array:

return 1 + np.random.normal(0, pd\_deviation\_sigma, size=size)

def \_\_calc\_uniformity\_normalized(bit\_vector: np.array) -> float:

unique, counts = np.unique(bit\_vector, return\_counts=True)

uniformity = 1 - 2 \* abs(0.5 - counts[1] / bit\_vector.size)

return uniformity

def \_\_calc\_uniformity(bit\_vector: np.array) -> float:

unique, counts = np.unique(bit\_vector, return\_counts=True)

uniformity = counts[1] / bit\_vector.size

return uniformity

def \_\_calc\_stable(responses: list) -> float:

print(responses)

s = np.add.reduce(responses)

print(s)

mask = (s != 0) & (s != len(responses))

print(mask)

# mask = s != len(responses)

unique, counts = np.unique(mask, return\_counts=True)

print(unique)

print(counts)

return 0 if len(counts) == 1 else 1 - counts[0] / len(s)

def \_\_calc\_reliability(responses: list) -> float:

ref = responses[0]

s = 0

m = len(responses)

for i in range(1, m):

s += distance.hamming(responses[i], ref)

return 1 - 1 / m \* s

def \_\_calc\_uniquness(fpga\_resps: list) -> float:

n = len(fpga\_resps)

s = 0

for i in range(0, n - 1):

for j in range(i + 1, n):

s += distance.hamming(fpga\_resps[i], fpga\_resps[j])

return 2 / (n \* (n - 1)) \* s

def read\_pd\_from\_file\_limited(filename: str) -> np.array:

with open(filename, 'r') as file:

lines = file.readlines()[:200]

result = np.zeros(len(lines), dtype=np.float32)

for i, l in enumerate(lines):

result[i] = float(l)

return result

def read\_pd\_from\_file(filename: str) -> np.array:

with open(filename, 'r') as file:

lines = file.readlines()

result = np.zeros(len(lines), dtype=np.float32)

for i, l in enumerate(lines):

result[i] = float(l)

return result