Оглавление

[Вступление 2](#_Toc493000363)

[С места в карьер! 3](#_Toc493000364)

[Метод Бокса-Мюллера (Box-Muller) 6](#_Toc493000365)

[Теоретическое обоснование: 6](#_Toc493000366)

[Имплементация генератора шума методом Бокса-Мюллера: 9](#_Toc493000367)

[Потребление ресурсов ПЛИС: 13](#_Toc493000368)

[Метод Уоллеса (Wallace) 14](#_Toc493000369)

[Теоретическое обоснование: 14](#_Toc493000370)

[Имплементация генератора шума методом Уоллеса: 16](#_Toc493000371)

[Потребление ресурсов ПЛИС: 20](#_Toc493000372)

[Метод Зиккурата (Ziggurat): 21](#_Toc493000373)

[Теоретическое обоснование: 21](#_Toc493000374)

[Имплементация генератора шума методом Зиккурата: 23](#_Toc493000375)

[Потребление ресурсов ПЛИС: 26](#_Toc493000376)

[Заключение 27](#_Toc493000377)

[Список литературы 28](#_Toc493000378)

# Вступление

Среди всех источников шума наиболее распространенным на практике и наиболее широко используемым в качестве модели случайного процесса является шум, описываемый нормальным (Гауссовским) распределением. Такой шум возникает в результате одновременного воздействия многих независимых случайных источников. Нормальное распределение отражает положения центральной предельной теоремы теории вероятностей, согласно которой случайная величина х, полученная суммированием статистически независимых случайных величин х1, х2, … хn с произвольными плотностями, имеет плотность, приближающуюся к нормальной, если n стремится к бесконечности. Типичным примером шума с нормальной плотностью является тепловой шум, обусловленный броуновским движением электронов в проводнике. Шум подобного типа принято называть белым шумом. Наибольший интерес при анализе систем представляет аддитивный белый гауссовский шум.

Аддитивный белый гауссовский шум (АБГШ, англ. AWGN) — вид мешающего воздействия в канале передачи информации. Характеризуется равномерной, то есть одинаковой на всех частотах спектральной плотностью мощности, нормально распределёнными временными значениями и аддитивным способом воздействия на сигнал. Наиболее распространённый вид шума, используемый для расчёта и моделирования систем радиосвязи.

* Аддитивный, потому что данный вид шума суммируется с полезным сигналом и статистически не зависим от сигнала.
* Название «белый» шум получил от белого света, который содержит электромагнитные волны частот всего видимого диапазона электромагнитного излучения.
* Гауссовский, потому что он имеет нормальное распределение во временной области с нулевым математическим ожиданием (μ = 0) и единичным отклонением (σ = 1).

В литературе описан широкий спектр генераторов случайных чисел Гаусса (GRNG). Все они используют хорошо понятые математические принципы, обычно включающие преобразования равномерных случайных чисел. Предполагая подходящую арифметику, GRNG можно в целом сконфигурировать для формирования случайных чисел достаточного качества для удовлетворения потребностей конкретной среды моделирования.

# С места в карьер!

Хочется начать отчет с конца. Хочется вместо описывания теоретических обоснований алгоритмов и их реализаций, представить сравнительные таблицы. В различных статьях сравнения разнятся (так как они сделаны для разного железа), но можно уловить основные тенденции и выбрать наиболее простые и производительные варианты.

Рассмотрим работу [1]. В ней представлена таблица, находящаяся ниже. В этой таблице:

* В первом столбце находятся названия методов/алгоритмов реализации генераторов шума;
* Во втором столбце показана скорость алгоритмов по отношению к полярному алгоритму (**Polar** [1969]) (количество операций на сгенерированное случайное число);
* В третьем столбце показано количество входных равномерных случайных чисел (U);
* В последнем столбце отражено количество констант, используемых в реализации (С);
* В остальных столбцах отражено наличие математических операций.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Speed** | **U** | **+** | **x** | **/** | **Cmp** | **()^0.5** | **Ln,exp,**  **trig** | **С** |
| **Wallace** (qual = 1) [2] | 6.41 | 0.001 | 10.02 | 1.50 | + | 1.51 | + |  | 9 |
| **Ziggurat** (2000) [3] | 4.29 | 1.04 | 1.10 | 1.07 |  | 2.07 |  | 0.001,  0.03,  0 | 388 |
| **Wallace** (qual = 4) [2] | 2.48 | 0.003 | 37.07 | 3.01 | + | 3.04 | + |  | 9 |
| **Monty Python** [4] | 1.61 | 1.30 | 0.88 | 1.96 |  | 2.57 |  | 0.03, 0,  0 | 16 |
| **PPND7** (ICDF) [5] | 1.16 | 1 | 8.15 | 7.40 | 1 | 1.45 | 0.15 | 0.15, 0,  0 | 26 |
| **Mixture-of-Triangles** [6] | 1.14 | 3 | 3 | 2 | 1 | 1 |  |  | 122 |
| **Polar** [1969] [7] | 1.00 | 1.27 | 1.91 | 3.27 | 1 | 1.27 | 1 | 1,  0,  0 | 4 |
| **Leva** (Ratio) [8] | 0.98 | 2.74 | 6.84 | 6.89 | 1 | 3.12 |  | 0.01, 0,  0 | 9 |
| **Marsaglia**-**Bray** [9] | 0.94 | 3.92 | 3.22 | 1.36 | 0.01 | 1.42 | 0.006 | 0.01, 0.05,  0 | 15 |
| **GRAND** [10] | 0.92 | 1.38 | 8.65 | 6.49 | 1.16 | 4.88 |  |  | 27 |
| **Box-Muller** [11] | 0.81 | 1 | n/a | 2 |  | 0 | 0.5 | 0.5,  0,  1 | 2 |
| **Ahrens-Dieter**  [12] | 0.78 | 1.02 | 4.55 | 4.04 | 1.5 | 4.51 | 0.5 | 0, 0.01,  0 | 20 |
| **Kinderman** (Ratio) [13] | 0.76 | 2.74 | 3.20 | 4.34 | 1.84 | 3.44 |  | 0.23, 0,  0 | 6 |
| **Hastings** (ICDF) [14] | 0.62 | 1 | 8 | 7 | 2 | 1 | 1 | 1,  0,  0 | 7 |
| **PPND16** (ICDF) [1988] [5] | 0.55 | 1 | 14.45 | 14.85 | 1 | 1.45 | 0.15 | 0.15, 0,  0 | 52 |
| **Central-Limit** (n = 12) | 0.39 | 12 | 12 |  |  |  |  |  | 1 |
| **CLT-Stretched** [15] | 0.35 | 12 | 17 | 8 |  |  |  |  | 5 |

Далее приведено статистическое качество генераторов, измеренное по критерию χ2 и High-Sigma, с одинаковой точностью, используя целое число для преобразования с плавающей точкой (Standard) и полностью случайное дробное (Full-Fraction) преобразование. Генераторы, проходящие тест χ2 для более чем 236 выборок, показаны с использованием «+». Там, где High-Sigma-тестирование становится в вычислительном отношении неосуществимым перед отказом генератора, точка, в которой тестирование остановлено, помечено знаком «+». Запись «n / a» указывает что тест или параметризация к этому конкретному генератору не применяются.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **χ2 - test (log2[n])** | | **High-Sigma Test** | |
| Standard | Full-Fraction | Standard | Full-Fraction |
| **Wallace** (qual = 1) [2] | + | n/a | 6+ | n/a |
| **Ziggurat** [3] | + | + | 8.15 | 17.4 |
| **Wallace** (qual = 4) [2] | + | n/a | n/a | n/a |
| **Monty Python** [4] | 34 | n/a | 8.27 | 14.88 |
| **PPND7** (ICDF) [5] | 34 | 34 | 4.11 | 12.64 |
| **Mixture-of-Triangles** [6] | 15 | + | 17.3 | 17.3 |
| **Polar** [7] | 26 | n/a | n/a | n/a |
| **Leva** (Ratio) [8] | 36 | + | 8.09 | 11.59 |
| **Marsaglia**-**Bray** [9] | 36 | + | 9.2 | 17+ |
| **GRAND** [10] | 29 | 30 | 5.25 | 12.64 |
| **Box-Muller** [11] | + | + | 7.91 | 17+ |
| **Ahrens-Dieter**  [12] | 35 | + | 4.11 | 13.7 |
| **Kinderman** (Ratio) [13] | 35 | + | 8.35 | 15.78 |
| **Hastings** (ICDF) [14] | 26 | 35 | 5.57 | 13.96 |
| **PPND16** (ICDF) [5] | + | + | 7.91 | 17+ |
| **Central-Limit** (n = 12) | 20 | n/a | 0.99 | n/a |
| **CLT-Stretched** [15] | 28 | n/a | 2.84 | n/a |

«Таблицы показывают, что метод Уоллеса является самым быстрым (но, как будет показано далее, может страдать от проблем с корреляцией); метод Зиггурата, второй по скорости, примерно на 33% медленнее, но не страдает от проблем с корреляцией. Таким образом, когда поддержание чрезвычайно высокого статистического качества является первоочередной задачей, но с учетом этого ограничения также желательна высокая скорость, метод Зиггурата может быть наиболее подходящим выбором. Если требования к качеству не столь строгие, а важна именно скорость, то использование Уоллеса может быть уместным. Одним из недостатков Зиггурата является большое количество констант (388 для одинарной точности), поэтому в средах, где количество ресурсов ограничено, могут быть также применены более простые методы, такие как полярные или GRAND.» Такими словами заканчивается статья [1].

В работе [16] автор предлагает сравнительную таблицу реализаций различных алгоритмов, реализованных на ПЛИС.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Design** | [17] | [18] | [19] | [20] | [21] | [22] | [16] |
| **Method** | Box-Muller | Box-Muller | Box-Muller | Box-Muller | Wallace | Ziggurat | Box-Muller +  C.L.T. |
| **Slices** | 437 | 480 | 534 | 1528 | 770 | 891 | 407 |
| **Block RAM** | 0.5 | 5 | 2 | 3 | 6 | 2 | 1.5 |
| **Multipliers** | N/A | 5 | 3 | 12 | 4 | 2 | 3 |
| **GRN Bitwidth** | 12 | 16 | 16 | 16 | 24 | 32 | 16 |
| **Repetition period** | 218 | 2190 | 288 | 264 | 232 | 288 | 2255 |
| **Tail Accuracy** | 4 σ | 4.8 σ | 6.6 σ | 8 σ | 7 σ | N/A | 6.6 σ |
| **Speed**  (M samples /s) | 25 | 245 | 440 | 468 | 155 | 168 | 460 |

Как видно из предоставленных сравнительных таблиц, а также судя по моему поиску, самые производительные алгоритмы на данные момент – это алгоритм Бокса-Мюллера, алгоритм Уоллеса, алгоритм Зиккурата. Эти три алгоритма реализуются на FPGA, не корреллируют между собой. Ниже будет сравнение этих трех алгоритмов.

Эти три алгоритма представляют собой частные примеры трех основных ветвей реализаций АБГШ:

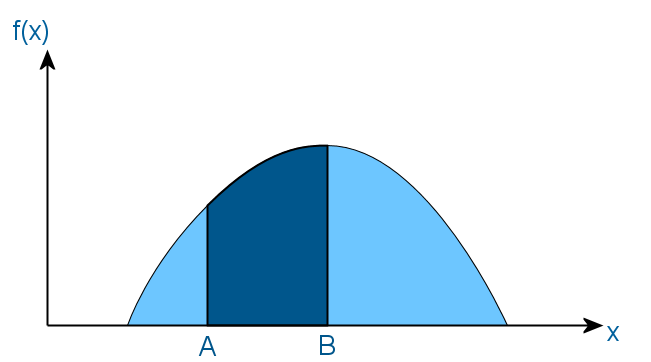
* Transformation method (Box-Muller, Central limit theorem, Piecewise Linear Approximation using Triangular Distributions, Monty Python),
* Rejection method (Ziggurat, Polar, Marsaglia-Bray, Ratio of Uniforms, Ahrens-Dieter Table-Free, GRAND, ),
* Recursive Method (Wallace).

# Метод Бокса-Мюллера (Box-Muller)

## Теоретическое обоснование:

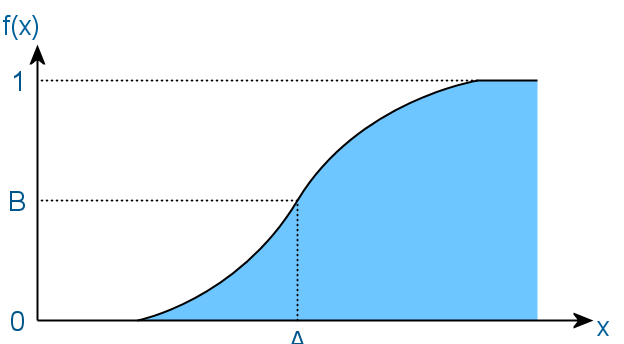
Преобразование Бокса—Мюллера — это метод моделирования стандартных нормально распределённых случайных величин.

Для начала стоит напомнить, что такое плотность вероятности, функция распределения случайной величины и обратная функция. Допустим, есть некая случайная величина, распределение которой задано функцией плотности f(x), имеющей следующий вид:



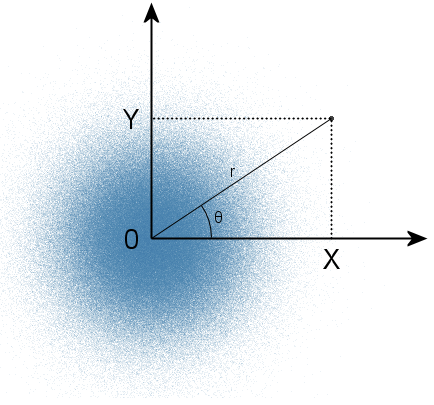
Основу всех методов преобразования равномерного распределения в любое другое составляет метод обратного преобразования. Работает он следующим образом: находится функция, обратная функции необходимого распределения, и в качестве аргумента передается в нее равномерно распределенная на отрезке (0, 1) случайная величина. На выходе получаем величину с требуемым распределением.

Это означает, что вероятность того, что значение данной случайной величины окажется в интервале (A, B), равняется площади затененной области. И как следствие, площадь всей закрашенной области должна равняться единице, так как в любом случае значение случайной величины попадет в область определения функции f.  
Функция распределения случайной величины является интегралом от функции плотности. И в данном случае ее примерный вид будет такой:



Тут смысл в том, что значение случайной величины будет меньше чем A с вероятностью B. И как следствие, функция никогда не убывает, а ее значения лежат в отрезке [0, 1].

Существует распределение **χ2** (распределение Пирсона), которое представляет собой распределение суммы квадратов k независимых нормальных случайных величин. И в случае, когда k = 2, это распределение является экспоненциальным. Это означает, что если у точки в прямоугольной системе координат будут случайные координаты X и Y, распределенные нормально, то после перевода этих координат в полярную систему (r, θ) квадрат радиуса (расстояния от начала координат до точки) будет распределен по экспоненциальному закону, так как квадрат радиуса — это сумма квадратов координат. Плотность распределения таких точек на плоскости будет выглядеть следующим образом:



Так как она равноценна во всех направлениях, угол θ будет иметь равномерное распределение в диапазоне от 0 до 2π. Справедливо и обратное: если задать точку в полярной системе координат с помощью двух независимых случайных величин (угла, распределенного равномерно, и радиуса, распределенного экспоненциально), то прямоугольные координаты этой точки будут являться независимыми нормальными случайными величинами. А экспоненциальное распределение из равномерного получить уже гораздо проще, с помощью того же метода обратного преобразования. В этом и заключается суть полярного метода Бокса-Мюллера.  
Теперь выведем формулы.

;

;

Для получения r и θ нужно сгенерировать две равномерно распределенные на отрезке (0, 1) случайные величины (назовем их u и v), распределение одной из которых (допустим v) необходимо преобразовать в экспоненциальное для получения радиуса. Функция экспоненциального распределения выглядит следующим образом: .

Обратная к ней функция: .

Так как равномерное распределение симметрично, то аналогично преобразование будет работать и с функцией:

*f(x) =*

Из формулы распределения **χ2** следует, что = 0,5. Подставим в эту функцию , v и получим квадрат радиуса, а затем и сам радиус:

;

;

Угол получим, растянув единичный отрезок до 2π:

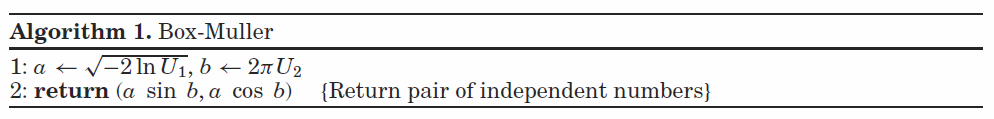
;

Теперь подставим r и θ в формулы и получим:

Эти формулы уже готовы к использованию. *x* и *y* будут независимы и распределены нормально с дисперсией 1 и математическим ожиданием 0. Чтобы получить распределение с другими характеристиками достаточно умножить результат функции на среднеквадратическое отклонение и прибавить математическое ожидание.

## Имплементация генератора шума методом Бокса-Мюллера:

Метод Бокса-Мюллера можно представить простым псевдокодом:



Все рассмотренные мною реализации схожи между собой по составу структурных блоков. Они имеют следующий вид и последовательность:

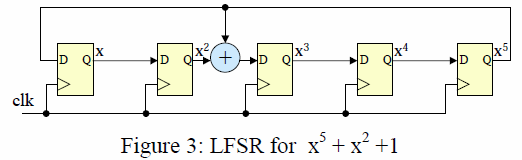
1. Формирование двух случайных чисел u0, u1;

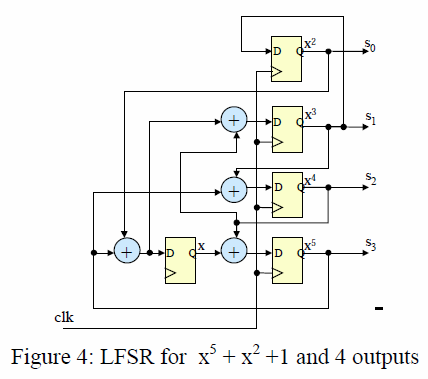
2. Вычисление и последующее вычисление

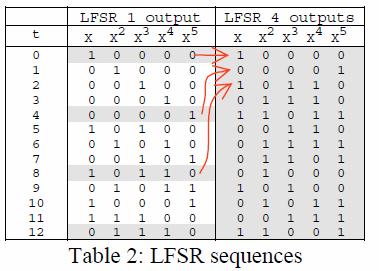
3. Вычисление и ;

4. Получение двух случайных величин и ;

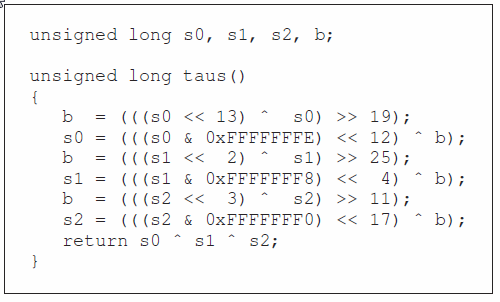
Формирование двух случайных величин разнится в различных источниках. В более ранней работе [23] этим занимался LFSR(Linear Feedback Shift Register). Этот LFSR имеет полином x5+x2+1. В этой же статье предлагается его усовершенствование:



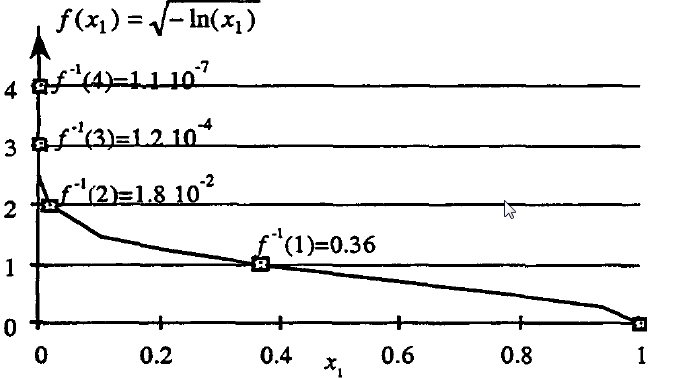




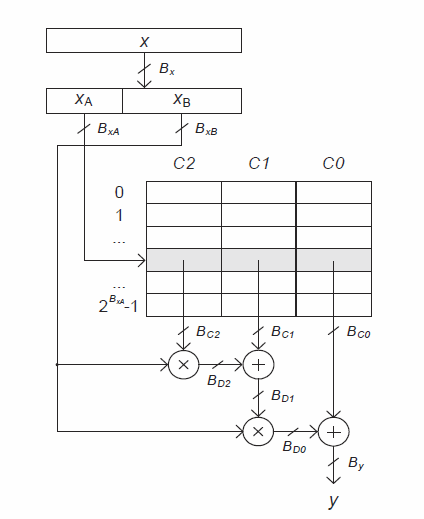
В более поздних работах, например в [24] используется Tausworthe URNG, который быстрее и занимает меньше площади. Кроме того, как пишет автор статьи, он обеспечивает превосходную случайность при оценке с использованием набора тестов случайных чисел Diehard. Реализация такого генератора на языке C показана на картинке ниже:



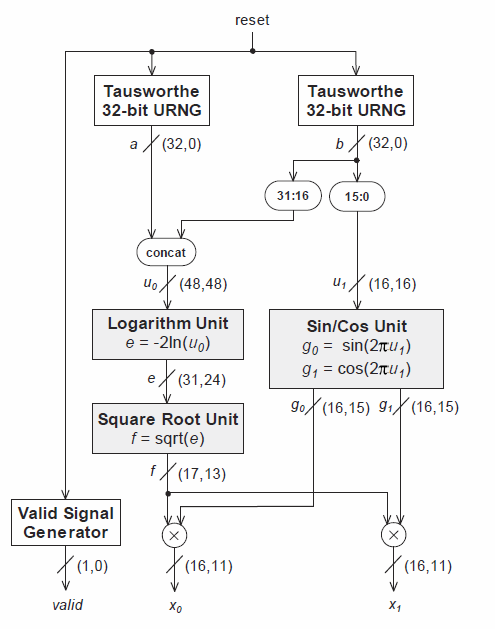
Вычисление логарифма и тригонометрических функций производится следующим образом: в буфер памяти записываются дискретные значения этих функции определенной точности. Адресом для ячеек являются сформированные генератором случайные величины. Процесс дискретизации этих функций также различен. Так, например, в [25] напрямую взяты значения функции (картинка ниже)

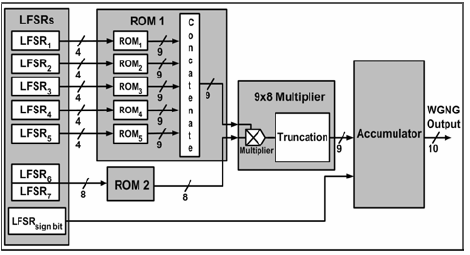


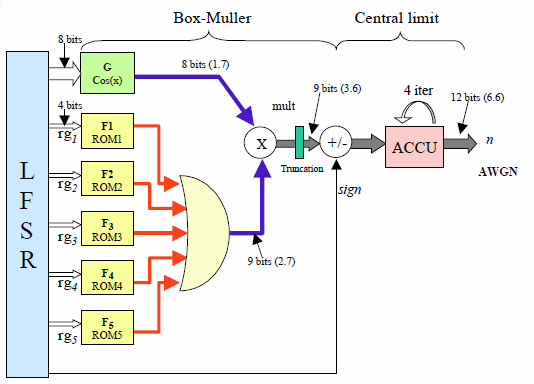
В [20] корень из логарифма и тригонометрические функции аппроксимированы полиномами, и в памяти хранятся значения полиномиальных коэффициентов. При получении этих коэффициентов происходит расчет функции. Пример показан на рисунке ниже:



После получения значения функций, значения перемножаются. Функциональные схемы различных реализаций представлены ниже:



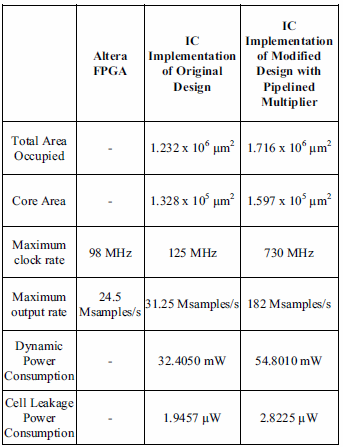




## Потребление ресурсов ПЛИС:

Для сравнения, ниже представлены распределения ресурсов ПЛИС для различных реализаций генератора шума методом Бокса-Мюллера:

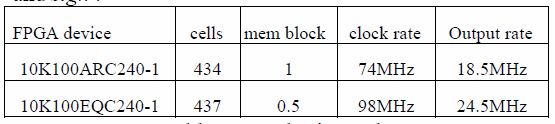
1) В работе [26]:



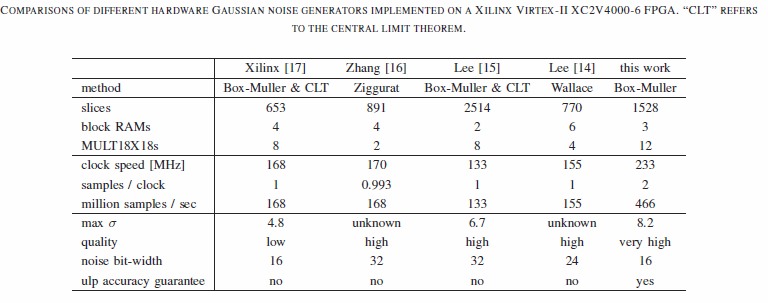
2) В работе [25]:



3) В работе [23]:



4) В работе [24]:

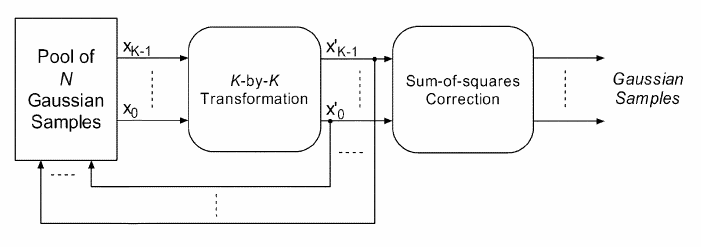


# Метод Уоллеса (Wallace)

## Теоретическое обоснование:

Генератор случайных чисел Уоллеса полагается на то, что линейные комбинации случайных чисел, распределенных по Гауссу, сами по себе распределены по Гауссову распределению.

Уоллес предлагает быстрый алгоритм генерации нормально распределенных псевдослучайных чисел, который генерирует нужные распределения напрямую, используя их свойства. Этот алгоритм особенно подходит для высокопроизводительной аппаратной реализации, поскольку не требует расчета сложных функций, таких как ,, (которые используются, например, в методе Бокса-Мюллера или Зиккурате). Схематическое представление метода Уоллеса показано на рисунке ниже.



Берется пул *N = KL* нормально распределенных случайных чисел из нормального распределения. Эти значения нормализуются так, что их среднеквадратичное значение равно единице. За *L* этапов преобразования K чисел, рассматривающийся как вектор *x*, преобразуются в K новых чисел преобразованием , где *A* - ортогональная матрица. Если исходные значения *K* распределены нормально, то и новые значения *K* будут распределены нормально. Кроме того, это преобразование сохраняет сумму квадратов. Процесс создания нового набора нормально распределенных случайных чисел называется «проходом». После «прохода» формируется набор новых Гауссовских случайных чисел. Поскольку в пуле данных есть *KL* переменных, в каждом проходе выполняются *L* шагов преобразования. *K-*размерныйвектор *x* умножается на ортогональную матрицу *A* при выполнении этапа преобразования.

Как утверждает Уоллес, желательно, чтобы любое значение в пуле в конечном счете вносило вклад в каждое значение в пуле, сформированное после нескольких проходов. В исходном методе Уоллеса старый пул рассматривают как массив *L*-по-*K*, хранящийся в строчном порядке, а новый проход рассматривается как массив *L*-по-*K*, хранящийся в порядке столбца. Следовательно, каждый проход эффективно переставляет значения в пуле. Если *L* нечетно, транспонирование достаточно для обеспечения возможного смешения значений. Однако, если *L* четно, одной транспозиции недостаточно.

Начальные значения в пуле нормированы так, что их среднеквадратичное значение равно единице. Поскольку A – ортогональная матрица, последующие проходы не изменяют сумму квадратов. Это было бы недостатком, так как если *x1, ... , xn* – являются независимыми выборками из нормального распределения, то мы ожидаем, что будет иметь хи-квадрат распределения . Чтобы преодолеть этот дефект, случайная величина из предыдущего пула используется для аппроксимации случайной выборки S из распределения . Вводится коэффициент масштабирования для обеспечения того, чтобы сумма квадратов значений в пуле составляла S.

Одной из проблем метода Уоллеса является корреляция использования предыдущих данных для создания новых. Это может быть проблематичным в случае реализаций с большими абсолютными значениями, лежащими в хвостах Гауссова распределения, поскольку каждое значение вносит *K* значений в последующий блок, *K2*-значений в следующий блок и т.д. с уменьшающимся влиянием. При правильном выборе параметров, таких как размер пула и преобразования, эффект этих корреляций может быть сведен к минимуму для заданного набора требований относительно количества выборок, точности хвоста и качества шума.

## Имплементация генератора шума методом Уоллеса:

В работе [21] представлена такая структурная схема имплементации метода Уоллеса:

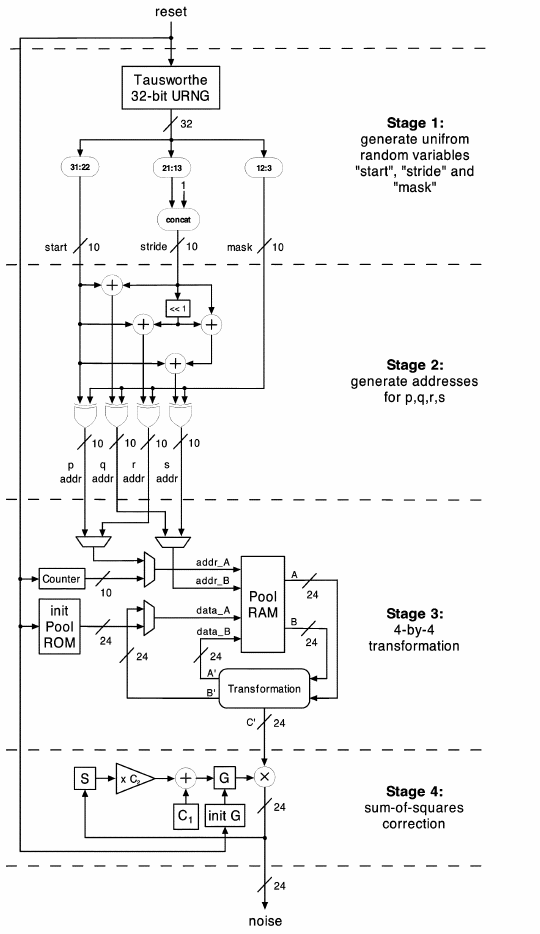


Схема состоит из 4 этапов:

*Этап 1:*

Этот этап включает в себя создание равномерно распределенных величин start, stride и mask. В этой работе также предлагается использование Tausworthe URNG, который обеспечивает лучшую случайность со меньшими аппаратными затратами. Используемый Tausworthe URNG объединяет три генератора случайных чисел на основе LFSR для получения улучшенных статистических свойств, генерирует 32-битное равномерное случайное число за такт и имеет период около 288. Поскольку мы используем пул размером 1024, каждая из трех генерируемых величин — 10-битная. Поскольку stride (шаг) всегда должен быть нечетным, мы объединяем (конкатенируем) получившееся значение с единицей после младшего значащего бита. Следовательно, для каждого прохода используются всего 29 бит для start, stride и mask. Остальные три бита остаются неиспользованными.

*Этап 2:*

Этот этап генерирует четыре адресных значения p, q, r, s из start, stride и mask. Эти адреса рассчитываются следующим образом:

p addr = start mask;

q addr = (start + stride) mask;

r addr = (start + stride x 2) mask;

s addr = (start + stride x 3) mask;

Умножение на два реализуется сдвигом влево, умножение на три осуществляется сдвигом влево, а затем суммированием (a x 3 = a x 2 + a). Эта схема адресации гарантирует, что корреляции между переменными становится минимальной.

*Этап 3:*

Этот этап включает в себя наиболее интересную задачу: эффективное выполнение фактической трансформации. Этот этап содержит «Pool RAM», который содержит пул из 1024 гауссовых случайных величин. Для реализации пула используется двухпортовая память. Поскольку каждая переменная в пуле составляет 24 бита, общий размер пула составляет 1024 \* 24 = 24576 бит. «init Pool ROM» и counter используются для инициализации пула, когда установлен сигнал сброса. Эта память — однопортовая и имеет тот же размер, что и пул. Содержимое этой памяти генерируется в программном обеспечении с использованием метода Box-Muller, а переменные нормализуются так, что их сумма квадратов равна единице.

Как и в оригинальной реализации Уоллеса, выбраны две ортогональные матрицы (матрицы Адамара):

A0 A1

Для данного набора из четырех значений p,q,r,s которые должны быть преобразованы, и с нашим выбором A0 и A1, новые значения p`,q`,r`,s` можно рассчитать из старых следующим образом:

p’ = p – t; q’ = t – q; r’ = t – r; s’ = t – s; (для матрицы A0);

p’ = t – p; q’ = t – q; r’ = t – r; s’ = t – s; (для матрицы A1);

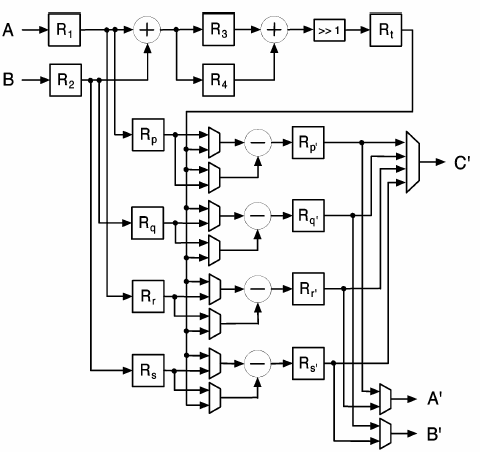
Где t =

На рисунке ниже показано, как выполняются шаги трансформации, описаные в двух формулах выше.t рассчитывается в три этапа

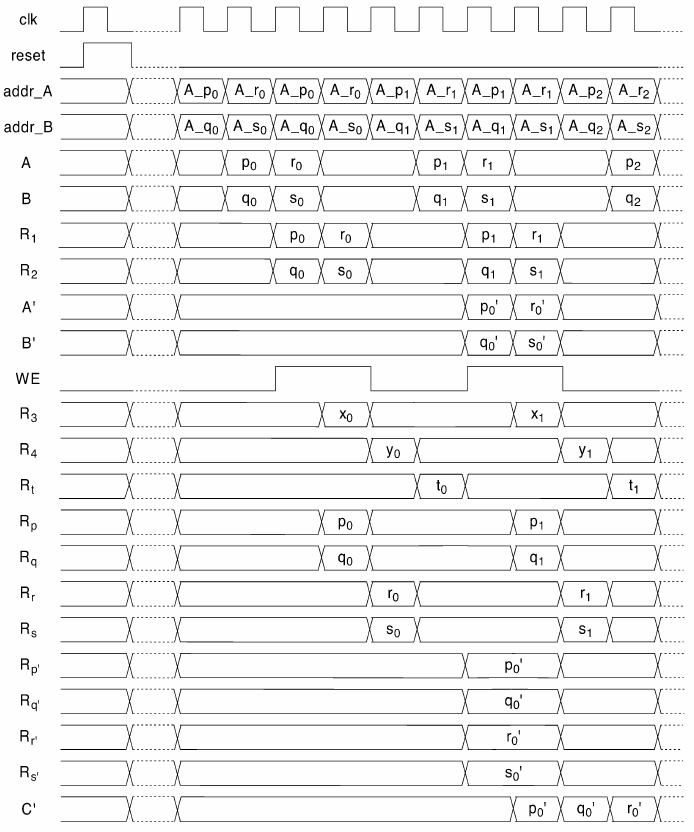
x = p + q;

y = r + s;

t = =



Временная диаграмма исполнения трансформации показана ниже:



*Этап 4:*

Этот этап выполняет исправление суммы квадратов, описанное ранее. Случайную величину S с приближенным распределением можно получить как:

, где х нормально распределенная величина, и для больших *N*. Следовательно, S можно вычислить, как

, где . Устанавим и .

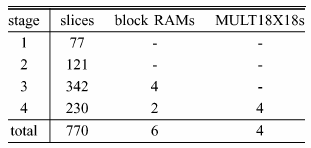
Образец шума C’, сгенеренный из схемы преобразования на *Этапе 3*, умножается на G, чтобы скорректировать сумму квадратов и, следовательно, конечный сэмпл шума. G получается как,

Поскольку C1 и C2 являются константами, они предварительно вычисляются в программно и сохраняются как константы в этом дизайне.

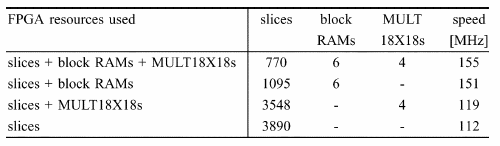
Перед проходом, S присваивается переменная из предыдущего прохода, а G обновляется. Для самого первого прохода, когда установлен сигнал сброса, G инициализируется в , где ts - сумма квадратов исходного пула.

## Потребление ресурсов ПЛИС:

Распределение ресурсов ПЛИС (XILINX VIRTEX-II XC2V4000-6 FPGA) при имплементации генератора шума методом Уоллеса по этапам(stage):



Распределение ресурсов ПЛИС (XILINX VIRTEX-II XC2V4000-6 FPGA)при имплементации генератора шума методом Уоллеса с использованием различных типов ресурсов:



# Метод Зиккурата (Ziggurat):

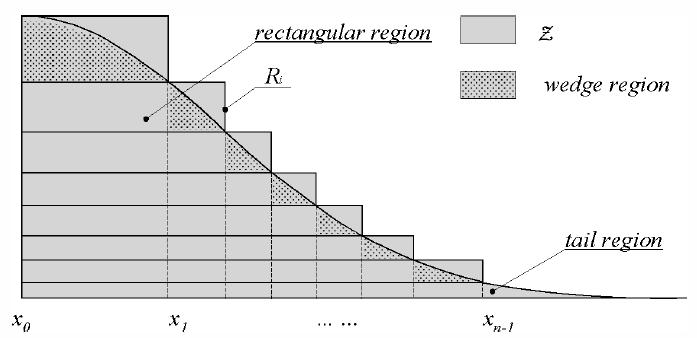
## Теоретическое обоснование:

Начнем с описания базового метода отклонения (Rejection Method). Пусть C - множество точек , лежащих под кривой с конечной площадью. Пусть *Z* - множество точек, содержащих *C* : . Основная идея метода отклонения: последовательный равномерный выбор случайных точек из Z, в случае, если точка попадает в *C*, в качестве требуемой случайно величины возвращается координата *x* этой точки. Плотность такого *x* будет *cf(x)*, где с - нормализующая константа, которая делает *cf(x)* требуемой плотности.

Тремя из основных критериев для выбора покрытия множества Z являются:

1. Простой и быстрый выбор случайной точки из Z;
2. Простое и быстрое принятие решения попадает ли случайная точка из *Z* в *C*;
3. , тогда эффективность процедуры отклонения тоже будет .

Метод Зиккурата удовлетворяет всем этим критериям более, чем любой другой общий известный метод. Грубая версия метода Зиккурата изображена на рисунке ниже:

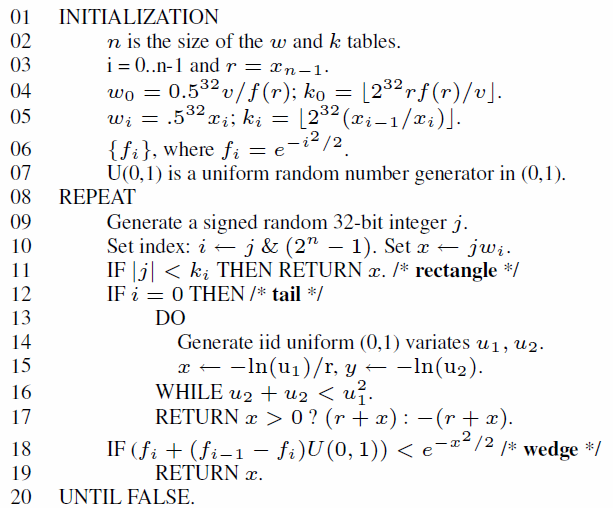


Здесь С - область под кривой , и *Z* - объединение 8 наборов равной площади *v*, 7 прямоугольников (rectangular region) и нижняя полоса, тянущаяся к бесконечности (tail region). (Для ясности выбрано только 8 наборов, но на практике их бывает 64, 128 или 256 наборов, объединение которых есть *Z*.) Кроме того, 7 из 8 наборов представляют собой прямоугольники, из которых легко получить случайную точку , и, кроме того, для этих прямоугольников легко решить, находится ли в C: если крайние правые координаты прямоугольников равны , а выбран прямоугольник Ri, *i > 0*, тогда x-координата случайной точки в Ri равна с *U*(0,1), а если , то случайная точка должна быть в C, что означает принятие *x* без генерации *y*.

И, наконец, легко получить случайную точку из базовой полосы (нижняя), так как база сама по себе является прямоугольником, примыкающим к хвосту, . Пусть *r* - крайний правый , так чтобы хвост был правее, . Мы можем генерировать *x* из базовой полосы следующим образом: порождать , причем U(0,1). Если возвращается x, в обратном случае возвращается *x* из хвоста (tail region). Это дает х из базового прямоугольника с требуемой вероятностью: . Обратите внимание, что код, предоставляемый для прямоугольной части базовой полосы, может быть такой же, как и для других прямоугольников: сформировать *x* и вернуть это *x* после успешного теста по магнитуде.

Для нормального хвоста метод Марсалья [27] предлагает: генерировать ; , пока , затем возвращается . Для экспоненциального хвоста .

Следующий псевдокод описывает полный алгоритм Зиккурата, обозначая генерацию в прямоугольной области (**rectangle**), области клина (**wedge**) и хвоста (**tail**):



## Имплементация генератора шума методом Зиккурата:

На рисунке ниже находится блочная диаграмма, показывающая основные компоненты генератора:

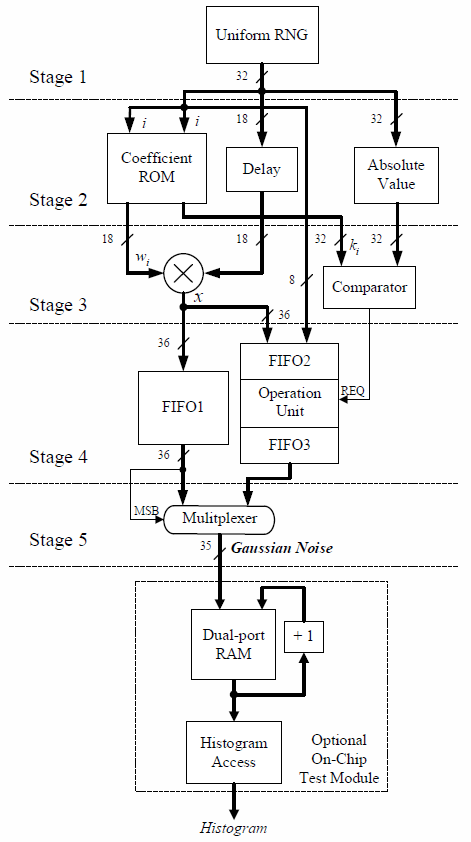
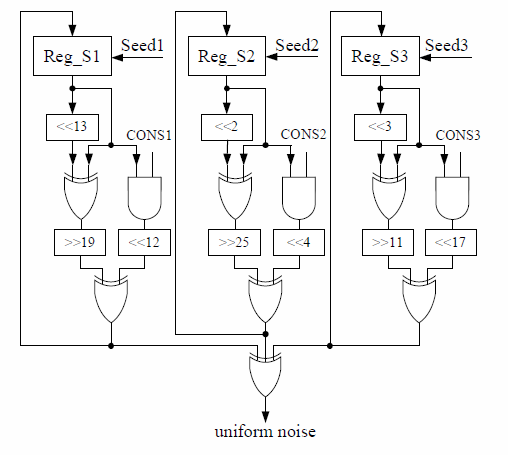


Схема содержит *несколько* *основных блоков*:

*Tausworthe Random number Generator:*

Linear FeedBack Shift Register (LFSR) часто используются для генерации однородных случайных чисел. В то время как традиционные LFSR являются достаточными для многих целей, генератор случайных чисел Tausworthe предлагает превосходную случайность с низкими аппаратными затратами, поэтому он предпочтительнее для приложений, в которых применяются чрезвычайно строгие стандарты качества шума [5, 10]. Tausworthe URNG объединяет три генератора случайных чисел на основе LFSR для того, чтобы получить улучшенные статистические свойства. Период генератора равен 288. На рисунке ниже показана аппаратная реализация RNG Tausworthe.



*Rectangular Region Datapath and FIFOs:*

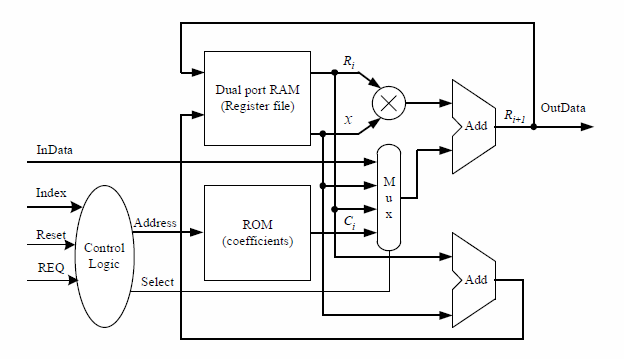
В псевдокоде (строка 9-11) генерируется случайная точка и происходит решение, находится ли она в прямоугольной области. Вычисление включает в себя умножение для получения *x* и сравнение |j| с ki. Если условие истинно, то старший бит (MSB) *x* устанавливается в единицу (иначе не используется), а *x* записывается на вывод прямоугольной области (FIFO1).

Если условие ложно, то случайная точка может соответствовать клину (wedge) или хвосту (tail). Затем маркер со сброшенным MSB записывается в FIFO1, а значение *x* записывается в FIFO2, который связывает прямоугольную область данных с Operation Unit. Эта буферизация служит для развязки вычислений прямоугольных областей, либо гауссовского случайного числа, либо маркера, указывающего, что клин/хвост записывается в FIFO1 каждый такт (пока он не станет заполненным). Operation Unit отвечает за обработку входов, полученных через FIFO2, и должен записывать свои выходы в FIFO3. Когда случайная точка x из хвоста или клина, MSB значения также устанавливается и записывается в FIFO3.

Когда выходное значение считывается из GRNG, из FIFO1 считывается число, а его MSB используется для управления мультиплексором. Если MSB установлен, используется значение из FIFO1, в противном случае возвращается прочитанное из FIFO3. Между тем, если MSB данных из FIFO3 установлен, конечным результатом является валидное случайное число, в противном случае это ошибочное испытание, для которого не генерируется достоверное случайное число. Обратите внимание, что MSB используется как маркер и не является частью случайного числа.

*Operation Unit (OU):*

Блок-схема *OU* показана на рисунке ниже. Она организована как регистровый файл и ALU, который включает в себя два сумматора и умножитель. Кроме того, ROM, используемое для хранения полиномиальных коэффициентов для вычисления элементарных функций ln и exp. *OU* упорядочивается через конечный автомат.



*Polynomial Evaluation*:

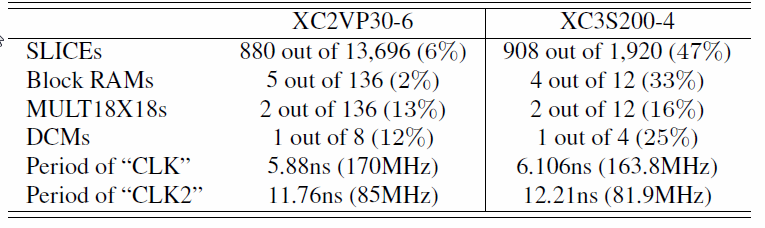
OU вычислят функции ln и exp в строках 19 и 22 псевдокода Зиккурата соответственно. Это, оказывается, является наиболее вычислительно дорогостоящей частью алгоритма Зиккурата, но, к счастью, поскольку они соответствуют областям клина и хвоста, которые встречаются с гораздо меньшей вероятностью, чем прямоугольные области, скорость, с которой приходится вычислять эти элементарные функции, не является настолько критической. В качестве примера, для n = 256, используемой в нашей реализации, объединенная вероятность отклонения клина, хвоста или образца составляет 1,5%, и поэтому скорость для этих секций должна составлять только 1,5% от скорости для прямоугольной области.

Прямая оценка полинома включает в себя оценку каждого монома индивидуально. Она принимает i умножений для каждого одночлена и n дополнений, в результате чего умножений для многочлена степени n. Правило Горнера, как правило, используется как в программной, так и в аппаратной реализации, обеспечивает лучшую численную стабильность и эффективность, умножая члены как: . Вычисление начинается с самых внутренних скобок, используя коэффициенты высшей степени и пробираясь наружу.

*Control Logic with State Machine:*

Здесь используется конечный автомат для реализации логики управления. Автомат начинается с состояния “S\_Start” и сначала ждет, пока сигнал REQ, который соответствует FIFO2, перестанет быть пустым. Когда сигнал REQ валиден, конечный автомат делает выбор между различными последовательностями состояний для областей клина и хвоста в зависимости от того, равен ли сигнал INDEX единицы или нулю соответственно. Конечный автомат генерирует адрес для коэффициентов, хранящихся в памяти, сигналы считывания и записи для регистрового файла и сигнал выбора мультиплексора. В случае клина (строки 22-23 в псевдокоде) функция оценивается в состоянии S\_EXP и сравнивается с в S\_WEDGE состоянии. Если INDEX = 0, мы оцениваем область хвоста (строки 16-21 в псевдокоде). Автомат будет вычислять два логарифма и выполнять сравнение в S\_LOG и S\_STATE состояниях соответственно. Вывод из клиновых или хвостовых вычислений записывается в FIFO3 в состоянии S\_END. После этой операции конечный автомат переходит в S\_NULL состояние, ожидая очередного сигнала REQ.

## Потребление ресурсов ПЛИС:



# Заключение

Как было сказано во вступлении, среди всех источников шума наиболее распространенным на практике и наиболее широко используемым в качестве модели случайного процесса является шум, описываемый нормальным (Гауссовским) распределением. Множество работ посвящено реализации генераторов шума различными математическими принципами. Из всей массы различных алгоритмов мною были выделены три основных: метода Бокса-Мюллера, метод Уоллеса, метод Зиккурата. Эти три алгоритма работают по совершенно разным математическим принципам, представляя основные подходы к реализациям генераторов шума: метод трансформации, метод отклонения, метод рекурсии. У каждого есть свои недостатки: у Бокса-Мюллера – высчитывание сложных математических функций, у Зиккурата – большое количество констант и долгий процесс генерации случайного числа из хвоста Гауссовского распределения, у Уоллеса – возможные корреляции между выходными значениями. Также эти алгоритмы обладают рядом преимуществ перед всеми остальными: Уоллес дают самые высокие показатели по частоте, Зиккурат дает наиболее высокую точность, Бокс-Мюллер, наверно, самый популярный и также наиболее прост в реализации.

# Список литературы

x

|  |  |
| --- | --- |
| 1. | Thomas D.B..L.W..L.P.H.W..A.V.J.D. Gaussian random number generators // ACM Comput. Surv., Vol. 4, No. 11, October 2007. pp. 1-38. |
| 2. | WALLACE C.S. Fast pseudorandom generators for normal and exponential variates // ACM Trans. Math. Softw., Vol. 22, No. 1, 1996. pp. 119-127. |
| 3. | MARSAGLIA G..T.W.W. The ziggurat method for generating random variables // J. Statis., Vol. 5, No. 8, 2000. pp. 1-7. |
| 4. | MARSAGLIA G.A.T.W.W. The Monty Python method for generating random variables // ACM Trans. Math. Softw., Vol. 24, No. 3, 1998. pp. 341-350. |
| 5. | WICHURA M.J. Algorithm AS 241: The percentage points of the normal distribution // Appl., Vol. 37, No. 3, 1988. pp. 477-484. |
| 6. | KABAL P. Generating Gaussian pseudo-random deviates // Tech. Rep., Department of Electrical and Computer Engineering, McGill University, 2000. |
| 7. | KNOP R. Remark on Algorithm 334 [g5]: normal random deviates. // Comm. ACM, Vol. 12, No. 5, 1969. P. 281. |
| 8. | LEVA J.L. A fast normal random number generator // ACM Trans. Math. Softw, Vol. 18, No. 4, 1992. pp. 449-453. |
| 9. | MARSAGLIA G.A.B.T.A. convenient method for generating normal variables // SIAM Rev., Vol. 6, No. 3, 1964. pp. 260-264. |
| 10. | BRENT R.P. Algorithm 488: A Gaussian pseudo-random number generator // Comm. ACM, Vol. 17, No. 12, 1974. pp. 704-706. |
| 11. | BOX G.E.P.A.M.E. A note on the generation of random normal deviates. // Annals Math.Stat., Vol. 29, No. 2, 1958. pp. 610-611. |
| 12. | AHRENS J.H.A.D.U. Efficient table-free sampling methods for the exponential, Cauchy, and normal distributions // Comm. ACM, Vol. 31, No. 11, 1988. |
| 13. | KINDERMAN A.J.A.M.J.F. Computer generation of random variables using the ratio of // ACM Trans. Math. Softw., Vol. 3, No. 3, 1977. pp. 257–260. |
| 14. | BOX G.E.P. A note on the generation of random normal deviates // Annals Math. Stat., Vol. 29, 1958a. pp. 610–611. |
| 15. | MULLER M.E. A comparison of methods for generating normal deviates on digital computers // J. ACM, Vol. 6, No. 3, 1959.. pp. 376-383. |
| 16. | Malik J.S. An efficient hardware implementation of high quality AWGN generator using Box-Muller method // The 11th International Symposium on Communication & Information Technologies (ISCIT 2011), October 2011. pp. 449-454. |
| 17. | Boutillon E. Design of High Speed AWGN Communication Channel Emulator // AICSP, 2003. |
| 18. | Additive White Gaussian Noise (AWGN) Core // v1.0, Xilinx Inc., 2002. |
| 19. | Alimohammad A. A Compact and Accurate Gaussian Variate Generator // IEEE Transactions on VLSI Systems, Vol. 16, No. 5, May 2008. |
| 20. | Lee D. A hardware Gaussian noise generator using the Box-Muller method and its error analysis // IEEE Trans. Comput, June 2006. |
| 21. | Lee D.U. A hardware Gaussian noise generator using the Wallace // IEEE Trans. Very Large Scale Integr. (VLSI) Syst., August 2005. |
| 22. | Zhang G. Ziggurat-based hardware Gaussian random number generator // Proc. IEEE Int. Conf. Field Program. Logic It’s, April 2005. |
| 23. | Danger J.L. // 7th IEEE International Conference on Electronicsm Circuits & Systemes, December 2001. |
| 24. | Lee D.U. A Hardware Gaussian Noise Generator Using the Box-Muller Method and Its Error Analysis // IEEE TRANSACTIONS ON COMPUTERS, Vol. 55, No. 6, 2006. |
| 25. | Ghazel A. DESIGN AND PERFORMANCE ANALYSIS OF A HIGH SPEED AWGN COMMUNICATION CHANNEL EMULATOR // IEEE, 2001. |
| 26. | Fung E. ASIC IMPLEMENTATION OF A HIGH SPEED WGNG // Department of Electrical and Computer Engineering, 2004. |
| 27. | Marsaglia G. Generating a variable from the tail of the normal distribution // Technometrics, Vol. 6, No. 101-102, 1964. |

x