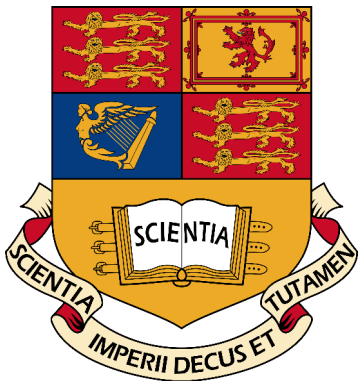
Imperial College London

Department of Electrical & Electronic Engineering



EIE Final Year Individual Project

Analysis of Random Numbers using FPGAs

Interim Report

Author:
Sol-Uh Park
CID: 00888669

Supervisor:
Dr. David Thomas

May 2, 2016

Abstract

With growing amounts of computationally intensive data being analysed in areas such as finance and engineering, there is growing research interest in faster hardware-accelerated Monte Carlo (MC) simulations. Faster MC simulations would in turn require faster random number generators (RNGs). Recent research has shown Field Programmable Gate Arrays (FPGAs) to provide an efficient platform for accelerating MC simulations and RNGs alike. However, for quality-checking RNGs, currently available RNG test suites are implemented in software and are not focused on throughput. Consequently, existing test suites currently present serious performance bottlenecks in the evaluation of the by several orders of magnitude faster FPGA RNGs. Given the growing need for better RNGs, there is strong research interest in efficient implementations of RNG test suites on the FPGA. This project will investigate the possibility of FPGA-accelerating RNG test suites, through exploring FPGA-optimised implementations of selected empirical statistical tests.

What is quality in this context?

I felt this is a little vague - it doesn't leave a concrete statement of what will be done, what the results will be, and how it will improve on the state of the art.

Self-propulsion : A- : good in terms of time management, with regular meetings and good note-taking. Also organised in terms of asking questions as they arise.

Technical progess: B : some aspects of technical work have been phrased by student, though with guidance.

Background research: B : Most decisions are well supported by background, in terms of the whys and whats. There are some citations found independently which are appropriate. Arguably there are some gaps in the background which provide high level overviews of the general before getting to the specific (e.g. trying to classify the problem more).

Critical Appraisal: B+ : the overall understanding shown from the report seems pretty good; the overall ideas come through, as well as most goals and constraints coming from the user.

Evaluation: B- : The evaluation planning contains the right components, but is too high level and not specific enough. It is not clear that the evaluation could be followed by another person and come to the same decision about whether it was successful or not. Clearer definition of what exactly the deliverables must do would help, such as the overal workflow, and how the user interacts with the system.

Planning: B- : The planning identifies risks, but some of the fallbacks lack specificity. The plan could also do a better job of explaining what components or outputs are delivered at each stage. The three implementation stages in the Gantt chart are not really fleshed out. What are the most important components (e.g. the histogram)? What is the true dependency graph between the different tasks, are there any blocking tasks?

The report itself is very text based. I would strongly suggest the use of figures/algorithms/maths as appropriate in order to more easily communicate the ideas. However, the structure of the report is fine (with some problems as noted), and I appreciate the heirarchical style and organisation.

Contents

1 Background

A Random Number Generator (RNG) is a software or hardware device that produces a sequence of reasonably unpredictable numbers. RNGs have diverse applications in man-made models where an element of unpredictability is a desired feature of the environment. Such applications are often found in areas such as cryptography, gambling, statistical experiments and computer simulations. When designing RNGs according to quality criteria such as period length, generation speed, "randomness" and required resources, the analysis of whether these criteria are satisfied is usually assisted through assessing the statistical properties of RNGs.

1.1 Why assess the Statistical Properties of RNGs?

Need to be careful about cryptographic versus non-crypt RNGs. What is the difference?

Misbehaving RNGs can have dangerous consequences. RNGs are used in diverse applications such as gambling and cryptography, where a traceable pattern in the sequence of generated numbers could pose financial and security risks to individuals and institutions. In computer simulations, insufficiently random inputs can bias the model, which can lead to dangerous misinterpretations.

The importance of quality-testing RNGs is highlighted in a notable case where Microsoft fixed Excel's default RNG in response to [1] finding that the Excel 2007 default RNG had been implemented erroneously. Among other findings, [1] observed that the RNG did not pass standard randomness tests, rendering Excel 2007 inappropriate for use in many statistical experiments and simulations. As observed by [2], Microsoft improved the RNG with better statistical properties in Excel 2010. This emphasises the importance of testing RNGs for their statistical properties prior to their deployment. Popular RNGs and their expected statistical shortcomings will be discussed later.

What is randomness? A short formal statement would help (plus possibly some comments on how achieveable it is in practise.

1.2 How to assess the Statistical Properties of RNGs?

A well-established approach is to apply theoretical, then empirical testing on the RNG.

1.2.1 Theoretical Statistical Testing

In theoretical statistical testing, the behaviour of the RNG's underlying algorithm is examined mathematically, usually through studying correlations within one period of the RNG's output sequence. Since the algorithms' underlying mathematical structure is assumed to remain unchanged irrespective of the developer's chosen platform for implementation, and given that there already exists extensive background literature on the theoretical statistical profile of most popular RNGs, theoretical statistical testing is considered to be outside of project scope.

What is a period?

The basic idea of RNGs (states, transition functions, period) should be briefly introduced before here.

1.2.2 Empirical Statistical Testing

In contrast, in empirical statistical testing, output stream sub-sequences are analysed using a battery of statistical tests. A test's goal is to find patterns indicating non-randomness according to a measure of what is thought to constitute non-randomness. An empirical test transforms a random sequence $u_0, u_1, ..., u_n$ to a real number Y, the test statistic, from which a pass or fail decision can be made. Popular empirical test packages include the Diehard battery of tests (1995) and the TestU01 library (2007). In this project, some of their statistical tests will be optimised and run in parallel in an FPGA-accelerated implementation, as will be discussed later.

A simple, but concrete, example of a test would have been helpful here (e.g. heads/tails).

1.2.3 Interpreting Empirical Test Results

Passing more empirical tests provides stronger confidence that the sequence under scrutiny is indeed random, and that its underlying RNG and its algorithm are good. Furthermore, [3] suggests that it could be generalised that a *bad* RNG is one that fails *simple* tests and a *good* RNG is one that passes *complicated* tests. However, classifying whether an RNG has passed a specific test is an ambiguous process as it is impossible to prove whether a sequence is perfectly random based on empirical testing alone. Instead, TestU01 and Diehard report p-values (probability values) to guide the user when deciding whether an RNG has almost certainly passed a certain test, or whether retesting is required with different parameters.

I like that as a summary.

It isn't immediately clear how p-values connect with Y mentioned above.

p-values are commonly known in the context of hypothesis testing, where conventionally a null hypothesis H_0 is rejected or accepted based on whether the reported p-value falls into a rejection area R, usually defined by a significance level threshold α. TestU01 however does not let the user specify R nor α. The TestU01 documentation [4] cites as reasons that in the context of RNG testing, the RNG sample size is usually huge and can be increased at will, making a fixed R and α inappropriate. Thus, unlike the standard procedure recommended in many statistical textbooks, where one should reject H_0 if and only if $Y \in R$, TestU01 simply reports the p-value. Both TestU01 and Diehard documentation [5] note that for perfectly random RNGs, the reported p-value would approximate a random variable $U(0, 1)$ under H_0, i.e. values between 0 and 1 should happen with about equal probability. An RNG is usually considered to fail a test if the p-values are clustered close to 0 or 1 for multiple iterations of the test in question. [3] defines the p-value as follows:

$$p = P[Y \geq y | H_0] \tag{1}$$

where y is the value taken by test statistic Y.

Summary of p-values in this context works quite well.

In this project, inspired by TestU01, each implemented test will report p-values using the definition above, as this methodology is well-established and easily understood by end-users of test suites.

1.3 Why design and analyse RNGs on FPGAs?

1.3.1 Monte Carlo simulations and FPGAs

What is the fundamental reason that MC takes so long?

Through its widespread use in solving computationally challenging problems, the Monte Carlo (MC) simulation has become an indispensable part of quantitative investigations in science, finance and engineering. The inherently high computation time however can span hours or days. Given that high run-time naturally translates to complexity constraints of the computational problem to be solved, this has driven demand for more efficient MC simulation implementations. Increases in efficiency would enable a multitude of new applications that rely on higher throughput. Recently, [6, 8, 7] have found that FPGAs show significant promise for accelerating MC simulations, primarily through the availability of fine-grain bit-wise operations, allowing high levels of parallelism using hundreds of on-chip arithmetic units, and exploiting other peculiarities of the FPGA platform. Furthermore, [8] asserts that one way of increasing MC simulation throughput is to formulate computational problems in "embarrassingly parallel" MC simulations, i.e. ones in which subsets of data are processed independently with very little communication whenever possible. FPGAs have been shown by [9] to be suitable for such use due to their high resource-efficiency.

1.3.2 Need for higher-throughput RNG implementations

MC simulations, especially in their embarrassingly parallel form, require resource-efficient, high-speed input streams of high-quality random numbers. In other words, slow RNGs are at risk of becoming performance bottlenecks. [9] has found that FPGA-based RNGs are faster and more resource-efficient than their their software-optimised counterparts. [8] has benchmarked RNG algorithms optimised for four different hardware platforms, finding that the FPGA shows the highest performance levels, whilst providing an order of magnitude higher performance per joule than any other platform. Furthermore, it is possible and convenient to feed the RNG output sequences directly into MC simulations on the same FPGA, making the platform attractive for such use.

1.3.3 Need for higher-throughput RNG test suites

FPGA-accelerated RNGs, like any RNG, have to be tested for their quality. However, their high performance can become a problem if test suites are several orders of magnitudes behind in terms of throughput and maximum sample size that can be tested for.

Furthermore, [12] notes that Diehard is not parameterisable, and consumes up to 2.5M 32-bit integers, whilst recent FPGA-based RNGs such as in [9] can generate 32-bit numbers at 800MHz+ without any optimisation. This would consume more than 320-times the Diehard maximum sample size every second. Whilst the newer TestU01 library can consume up to 2^{38} 32-bit integers, this still implies an analysis of only 6 minutes worth of 800MHz random numbers, whilst [8] notes that in highly parallel ar-

Good motivation!

chitectures, RNGs with periods ranging larger than 2^{160} are of research interest, which is several orders of magnitude larger than TestU01 constraints currently allow for empirical testing. In addition, interfacing between FPGA and software is likely to be slower than directly pipelining and parallelising the test suites on-chip, such that additional performance bottlenecks can occur when trying to analyse hardware-generated numbers in software. Given the aforementioned need for resource-efficient, high-speed, high-quality RNGs, there is growing interest towards testing FPGA-accelerated RNGs more efficiently, hence the importance of analysing FPGA-generated random numbers directly on the FPGA.

I think you're over-selling the "growing interest" part. It is of interest in a particular community in a particular way.

1.4 Related Work

As inspiration for the RNG tests to be implemented on the FPGA, the test suites TestU01 and Diehard have been used, in addition to existing literature from Jansson [10] and Knuth [11]. The selected tests and their specifications and modifications will be discussed later. Other than software test suites and academic literature, there currently exists no directly related work on the implementation of RNG tests on FPGAs. Whilst there have been some attempts to address test suite throughput issues in software, e.g. with [13] using multi-threading in software to achieve a 4x speedup of the TestU01 library, this is still several orders of magnitude too slow for faster hardware RNGs.

Good ref.

2 Project Specification

We're missing a re-statement of the goals of the project. Now that we know about the background, remind us what you're trying to do, and connect it to what we learnt in the background.

2.1 Target Platform

In this project, the ZedBoard Zynq-7000 ARM/FPGA SoC Development Board is used to implement the RNGs and their empirical tests. This FPGA board was chosen a) due to its useful hardware specifications such as the embedded ARM Cortex A-9 processor controlling a 512Mb DDR3 SDRAM module, b) due to its integrated software support for creating IP cores using high level synthesis, as discussed below.

For the synthesis and simulation of the tests, the Vivado Design Suite will be used alongside Vivado HLS which is a subset of the suite. Vivado HLS uses high-level synthesis to translate C++ specifications directly into RTL. Whilst Verilog/VHDL implementations can be fine-tuned for better performance, its high implementation time is seen as inappropriate for the tight project time frame. In fact, Vivado HLS comes with optimisation commands which can achieve speedups appropriate enough for this FPGA-accelerated implementation. These commands can be directly embedded in C++ code and can be specified granularly at function- or variable-level, before the code is synthesised. The use of optimisation techniques to speed up the implementation is a main focus in this project.

It would be good to see a work-flow diagram. What comes from the user, who compiles what, how are things run, what is the final output?

2.2 RNG Types within Project Scope

Whilst applications in areas such as cryptography and gambling may at times view the sequence's unpredictability as more important than its speed of generation, in contrast, in MC simulations, fast streams of resource-efficiently generated, "unpredictable enough" random numbers are desired. In this project, the client wishes to test RNGs designed to be used in MC simulations. Using this information, the RNGs under consideration in this project can be limited to the following types.

These concepts arguably should have been laid out in the background, and then here you state which ones are in scope.

2.2.1 Pseudorandom and Deterministic RNGs

MC simulations need to exhibit the property of simulation repeatability, i.e. for identical initial conditions, the simulation should always yield the same results. In order to satisfy this requirement, the FPGA-accelerated RNGs tested, implemented and discussed in this project will be pseudorandom and deterministic, i.e. for identical seed values, the RNGs should always output identical number sequences. In this project, a user-inputted seed value will be acknowledged by the test bench, and reported back to the user in an output data format that will be specified later in the report.

2.2.2 Uniform RNGs

RNGs can be classified into uniform RNGs and non-uniform RNGs. Uniform RNGs generate numbers which approximately occur with identical frequency over its defined range of possible output values. Numbers generated by non-uniform RNGs in contrast follow different frequency distributions such as Gaussian or exponential distributions. The standard approach for implementing a non-uniform RNG is to take a uniform RNG and use mathematical methods to plot its output to a desired non-uniform distribution. [8] has shown that RNGs on the FPGA platform achieve their highest performance for optimised uniform RNGs in particular, with a throughput higher by several orders of magnitude compared to optimised Gaussian or exponential RNGs. Given this finding, coupled with the tight project time frame, only uniform RNGs are within project scope.

2.2.3 32-Bit Unsigned Integer RNGs

Popular RNG test suites, including Diehard and TestU01, usually evaluate 32-bit RNGs by default. Related work also frequently reports RNG performance metrics for 32-bit numbers. For performance benchmarking purposes, this project will likewise implement tests for 32-bit integer RNGs. For simplicity's sake during project implementation stage, the range of integers will be considered to be unsigned. For true RNGs, this would imply that given a long enough period, every number between zero and 4,294,967,295 ($2^{32} - 1$) would occur at some point in its sequence.

2.3 Empirical Tests under Consideration

Given the FPGA's fine-grain bit-wise computation possibilities, the approach used in this project is to make use of this peculiarity through base conversion: Given that all 32-bit unsigned integers from 0 to 4,294,967,295 inclusive can alternatively be represented in base-2, each of the constituent bits 0 to 31 will be analysed for certain randomness properties in the tests under consideration.

More accurately, each test will compare a theoretical distribution of some randomness figure to the measured distribution of this figure, thus making it possible to compute the deviation (amount of error) between the two. Determining whether the deviation is likely to have occurred by chance can be done through chi-squared tests, which thus form the main building block for each test under consideration.

2.3.1 Chi-Squared Test

The test statistic Y is used to determine the likelihood of an observed frequency distribution being consistent with its theoretical distribution. The observations must be independent. Observations are grouped into k categories and the number of generated integers (sample size) is denoted by n. Then, Y is given by:

$$Y = \sum_{s=0}^{k-1} \frac{(Y_s - np_s)^2}{np_s} \tag{2}$$

where Y_s is the observed number of 1s in category s and p_s is the expected probability of a 1 occurring in category s. The following tests provide different Y_s and p_s according to the randomness property investigated, from which a different Y is computed. Through the use of the number of degrees of freedom v, the p-value can be computed using the chi-squared cumulative distribution function. The relevant distributions will be pre-computed in software to make p-values form part of the reporting back to the user in a file format which will be specified later.

It would make sense to explicitly give p in terms of Y and the Chi-squared distribution,

2.3.2 Frequency Test (Basic Implementation)

The motivation behind frequency testing is to evaluate whether the deviation between observed and expected distribution of 1s for each of the 32 bits is likely to have occurred by chance. Thus, Y is obtained through setting $k = 32$ and $p_s = \frac{1}{2}$ for all s, given a fixed 50% chance of a 1 occurring. The number of degrees of freedom is given by $v = k - 1 = 31$, from which the p-value is computed.

2.3.3 Serial Test (Intermediate Implementation)

This test checks whether the observed distribution of n-tuple successive numbers is approximately uniformly distributed as expected. In the simplest case for $n = 2$, the 2-tuples can be categorised into the four possible buckets $00, 01, 10, 11$. Note that in order to satisfy the chi-squared test's assumption of independent observations, the n-tuples must be non-overlapping.

For each of the 32 bits, there are 4 buckets. Thus, $k = 32 \times 4 = 128$ and $v = k - 1 = 127$. Given that each bit has a 50% chance of a 1 occurring, irrespective of the values of any of the other 31 bits, each tuple is expected to be uniformly distributed with $p_s = \frac{1}{k} = \frac{1}{128}$ for all tuples s. Using k, p_s and v, first Y and then the p-value can be evaluated, similarly to the frequency test. The serial test may be scaled to triples, etc if project time constraints allow for their exploration.

2.3.4 Gap Test (Intermediate Implementation)

This test checks whether the observed distribution of gaps is distributed as expected. A gap is defined as the distance between two 1s, e.g. the bitstream 10001 implies a gap of size 3 between the two successive 1s. Gaps may have the values $0, 1, 2, ..., f, ...$ and each gap of size s will be counted in a classification category, where the classification category f counts all gaps of size larger or equal to f. Hence, $0 \leq s \leq f$ which implies that $k = f + 1$. Denote the probability that the gap equals s by p_s. For base-2 numbers, p_s is given by: $p_s = \left(\frac{1}{2}\right)^s \times \frac{1}{2}$ where $(r = 0, 1, ..., n - 1)$.

Using k and p_s, Y can be computed. Using $v = f - 1$, the p-value can be computed.

Is this another chi-squared stat then?

2.3.5 Count the 1s (Advanced Implementation)

This test counts the number of 1s in the 32 bits of each generated number. This count figure (from 0 to 32 inclusive) is then converted into a "letter", and 5 successive letters are converted into a "word". The goal is to evaluate whether the deviation between observed and expected distribution of the words is likely to have occurred by chance.

Given that words consist of 5 consecutive letters and given that there are 33 different letters, $k = 33^5 = 39135393$. Since for uniform RNGs, every bit has a constant 50% probability of a 1 occurring, the probability of each letter α occurring is given by: $p_\alpha = \frac{\binom{32}{\alpha}}{2^{32}}$ for $0 \leq \alpha \leq 32$. Then, the probability of each word s occurring is given by $p_s = p_{\alpha 1} \times p_{\alpha 2} \times p_{\alpha 3} \times p_{\alpha 4} \times p_{\alpha 5}$ for $0 \leq s < 33^5$. Again, $v = k - 1 = 33^5 - 1$. Using k and p_s, Y can be evaluated. The p-value is obtained using Y and v.

2.3.6 Poker Test (Advanced Implementation)

The classical poker test considers five successive integers and checks whether the observed quintuple frequencies correspond to the expected distribution. Inspired by poker hands consisting of five cards, the following integer combinations can occur: All different (abcde), one pair (aabcd) two pairs (aabbc) three of a kind (aaabc), four of a kind (aaaaa) and poker (aaaaa).

In this project, given that the numbers under consideration are 32 bit numbers, the test will be modified such that hands of 4 integers will be used. 32 bits are more readily divisible by 4, which allows for a more efficient implementation of a 4-card test compared to a 5-card one. To model hands of 4 different integers, 4 pairs of 2 bits will be grouped together. This results in $\frac{32}{4\times 2} = 4$ hands for chi-squared analysis, with the chance of each number occurring equaling $\frac{1}{4}$. [14] computes the 4-card probabilities as:

- Four of a kind: 0.001
- Three of a kind: 0.036
- Two pairs: 0.027
- One pair: 0.432
- All different: 0.504

Obtaining $k = 4 \times 5 = 20$ and $v = k - 1 = 19$, and using the above probability values as p_s values for $0 \leq s \leq 19$, it is possible to compute Y to obtain the p-value.

The descriptions of the tests are fine, but you could have shown more of the commonality. In what ways are they similar, and can they be expressed in terms of shared components?

2.4 RNGs under Consideration

In the following, some popular RNGs will be introduced in ascending order of quality, which are planned to be implemented and tested for in this project to demonstrate the failing of different combinations of tests.

Constant Value: One of the worst RNGs is one returning a constant value throughout the specified sample size. This number generator is expected to fail every test.

Simple Counter: A simple counter outputs all possible 32-bit unsigned integer values in ascending order, and resets to zero upon reaching the final possible value. This RNG is expected to pass the frequency test and the count the 1s test, but is expected to fail the serial test and the gap test, as the ascending order number stream will obscure tuple distributions whilst keeping bit gaps constant for each bit s.

Gray Code Counter: Gray code is a numeral system where two successive numbers differ in only one bit. The RNG is expected to fail the serial test, since only one

bit changes at a time, obscuring the tuple distribution.

RANDU: RANDU is an RNG widely used in the 1970s despite its bad randomness properties. Its generated numbers corresponding to the recurrence:

$$V_{j+1} = 65539 \times V_j \bmod 2^{31} \tag{3}$$

where V_0 is an odd number. The generated numbers are uniformly distributed, however, the generator produces only odd numbers. The generator is expected to fail basic tests.

Tausworthe Generator: A Tausworthe generator is an RNG given by the modulo 2 recurrence:

$$V_i = (a_1 V_{i-1} + a_2 V_{i-2} + ... + a_n V_{i-n}) \bmod 2 \tag{4}$$

where $a_i (i = 1, 2, ..., n)$ are either 1 or 0. This generator is expected to pass all tests.

Isn't a goal also to test user-code? Where can you get an "advanced" RNG from?

2.5 Project Goals

Maximum Sample Size: In TestU01, the maximum sample size permitted is 2^{38}. It is thus envisaged that in this project, as a minimum requirement, 2^{38} numbers can be inputted. Once this milestone has been reached, a goal sample size of 2^{44} numbers will be aimed for.

Throughput: As a bare minimum, the implementation should work correctly and as expected, irrespective of how many clock cycles are consumed for sample numbers. Once a minimum implementation has been achieved, the goal is to process one sample every clock cycle.

Tests to be implemented: As a minimum requirement, a fully working frequency test will be aimed for. Once this milestone has been reached, the intermediary goal is to have all intermediate tests run in parallel. The advanced goal is to have all four tests running in parallel.

If felt this is a little too vague. It covers some functional requirements, but not others. How is the user's function specified? What comes out at the end? What about if the test is interrupted? How do you actually test that the generator is working properly at high throughput?

3 Implementation Plan

For this section, a complementary weekly-view Gantt chart is included in Appendix A.

3.1 Completed Project Stages

At this point in the project, the Vivado design suite's functionalities have been explored through compiling Vivado HLS example projects. In addition, through official and third-party laboratory tutorials available online, peculiarities of Vivado HLS such as its FPGA-optimisation techniques, how hardware can be modeled using C++ and directive commands have been explored. Furthermore, the mathematical background behind RNGs and the chosen tests has been established.

Top-down, the design is expected to be implemented in a single HLS core for sake of simplicity. As input, the RNG and the test bench will acknowledge a user-defined sample size and a seed value. Each test hardware module is expected to output the frequency distribution of each chi-squared category of the respective test in the form of histograms. Using this distribution data, the p-values can be computed either in hardware or software, which is unclear at this stage. The reporting will be done in a .csv file output format. Inspired by TestU01, it is envisaged that test results will be reported in a .csv format similar to the example below:

Sample size	1000000000
Seed	153295203
---	---
Test	p-value
---	---
Frequency	0.999985
Serial	0.125326
Gap	0.164346
Count the 1s	0.781293
Poker	0.001981

This is not a CSV format. Also, it probably needs a bit more information, for example what is the RNG being tested? Also, might it not make sense to also output the raw test statistic (or is there a reason not to)? It might also be useful to put out the raw histograms as well, for archival purposes.

Figure 1: Example .csv output format

If additional features are implemented (as discussed in the next section), the .csv output format will be adapted accordingly. Parts of the algorithm which are less suitable or not necessary for hardware acceleration, including resource-intensive computations for p-values from chi-squared distributions, will be moved to software whenever possible, and software will be assumed to be all-knowing about the properties of the histograms (size, what data represents, etc.) such that RNG pass/fail decisions can be made by the user whilst keeping hardware at a minimum complexity.

3.2 Next Project Stages

The next step is to adapt a basic implementation consisting of a basic RNG and the frequency test. Current uncertainties about how the architecture is to be pinned with FPGA components, and how hardware-software interfacing works in Vivado are expected to be resolved during this stage. Evaluating whether frequency test and its RNG work as intended is expected to be done through creating testbenches implemented in C++ in Vivado HLS, which allow for testing the correctness of C++ code prior to the code's synthesis and deployment on the FPGA. The hardware module for chi-squared testing developed as part of the frequency test is expected to be reused in subsequent tests, ideally reducing implementation time for each additional test.

Good point about reuse, but making dependencies explicit would have helped.

Next, the advanced RNGs and tests will be implemented in ascending order of complexity. How many will be implemented is subject to how much time is left at each implementation stage, as will be discussed. After the advanced implementation, should time remain, the possibility of checkpointing the long RNG tests will be explored, such that tests may report preliminary results while the tests continue to run, thus providing useful real-time feedback for the end-user.

3.3 Assessing Anticipated Risks

The project milestones may not be reached on time due to unforeseen circumstances: Some unknown unknown problems are likely to occur in the implementation stages, some of which may slow down the project significantly. Possible issues may occur in areas including, but not limited to, interfacing between FPGA and software, interfacing between hardware modules and debugging C++ code. Should time constraints manifest themselves during the test implementation stages, it is foreseen that as a fall-back, the implementation of the most advanced tests and RNGs will be relinquished. Progress will be checked each week to establish whether sacrifices have to be made.

These are definitely risks, and the fall-backs make sense, but the only mitigation mentioned is re-active, rather than pro-active.

Furthermore, at this stage in the project, it is uncertain whether the implementation of every mentioned test is feasible, since there are risks associated with running out of available resources on the FPGA. For instance, for the count the 1s test mentioned previously, the many chi-squared buckets needed may not be feasible for direct implementation on the FPGA, due to limitations on the number of look-up tables. If need be, impractical tests will be adapted in their methodology to suit the FPGA platform, similarly to the adaptation of the classical poker test as discussed previously. If this is unfeasible, the impractical tests may be replaced with more practical ones.

Good risk identification, but the mitigation is a bit vague.

In evaluation stage, as will be discussed in more detail in the next section, special care should be taken to avoid preventable project slowdowns such as preventable power outages causing the FPGA to crash halfway through a test lasting multiple hours.

4 Evaluation Plan

In order to evaluate whether the FPGA-implemented tests are working as intended, each RNG will be run through the parallelised tests at least once in real-time. Assuming that one sample is processed every clock cycle at 100MHz as supported by the FPGA board, the testing is expected to take two full days for 2^{44} samples ($\frac{2^{44}}{100M*60*60*24} = 2$ days), which is the goal maximum sample size. RNGs which have a period which is several orders of magnitude smaller than 2^{44}, such as the constant value, simple counter and Gray counter RNGs, may be tested for a significantly smaller sample size than the maximum allowed. This is because in these cases, run-times significantly longer than the period can be considered as not very meaningful for assessing RNG quality.

The test results obtained will be compared against the expected test results. As aforementioned, for multiple iterations of the tests, p-values are expected to be uniformly distributed between 0 and 1 and an RNG is likely to have failed tests when p-values repeatedly cluster close to 0 or 1 for multiple test iterations. Thus, the test results obtained can fall into one of three categories: a) definitely pass, b) definitely fail or, for ambiguous p-values, c) the "grey zone". Should the p-values fall suspiciously close to 0 or 1 for some tests on the first trial of an RNG, then the tests will be re-run with different parameters to achieve a higher level of confidence when categorising whether RNGs have passed a test. Furthermore, if possible with regards to the sample size constraints and if necessary, using a similar methodology, results can be obtained from TestU01 or Diehard (with modifications made to the tests if necessary). These will be compared against the results from the FPGA implementation, if necessary. Given that RNGs are deterministic, given identical seeds and sample size, the p-values should be similarly categorised into a) b) or c) for FPGA tests and software tests alike.

This is a bit weak in terms of verification. A clear statement "the p-value should be the same from both hardware and software" would help, unless this is not the expectation (it might not be).

5 Summary

This interim report has covered a number of fundamental concepts in RNG testing and how it relates to the FPGA, providing the theoretical framework for the implementation and simulation of RNG empirical statistical tests on the FPGA. Given the growing need for analysing very long streams of random numbers at high speeds on the FPGA for Monte Carlo simulations, this welcomes an implementation directly on the FPGA. RNG tests compare expected and observed frequency distributions of some randomness figure, from which p-values are computed which, should they repeatedly cluster suspiciously close to 0 or 1, indicate a failed test. The project goals of processing more than 2^{38} samples at a throughput of one sample per clock cycle are planned to be achieved as follows: The tests will be run in parallel whilst some of the tests will be modified to better exploit FPGA techniques, and their underlying C++ code will be optimised through Vivado implementation directives. It is envisaged that as many tests will be implemented in ascending order of difficulty as the project timeline allows.

6 References

References

[1] B.D. McCullough, David A. Heiser. *On the accuracy of statistical procedures in Microsoft Excel 2007*. Computational Statistics and Data Analysis 52 (2008) 45704578.

[2] Guy Mélard. *On the accuracy of statistical procedures in Microsoft Excel 2010*. Comput Stat (2014) 29:10951128.

[3] Pierre L'Ecuyer, Richard Simard. *TestU01: A C Library for Empirical Testing of Random Number Generators*. ACM Transactions on Mathematical Software, Vol. 33, No. 4, Article 22, Publication date: August 2007.

[4] Pierre L'Ecuyer, Richard Simard. *TestU01 A Software Library in ANSI C for Empirical Testing of Random Number Generators User's guide, detailed version.*

[5] G Marsaglia. *DIEHARD: a battery of tests of randomness (1996).* `http://stat.fsu.edu/geo/diehard.html`

[6] De Schryver, Christian, Ivan Shcherbakov, Frank Kienle, Norbert Wehn, Henning Marxen, Anton Kostiuk, and Ralf Korn.*An energy efficient FPGA accelerator for Monte Carlo option pricing with the Heston model*. Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on, pp. 468-474. IEEE, 2011.

[7] Phillip J. Kinsman, Nicola Nicolici. *NoC-Based FPGA Acceleration for Monte Carlo Simulations with Applications to SPECT Imaging*. Computers, IEEE Transactions on 62, no. 3 (2013): 524-535.

[8] David B. Thomas, Lee Howes, Wayne Luk. *A Comparison of CPUs, GPUs, FPGAs, and Massively Parallel Processor Arrays for Random Number Generation* Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, pp. 63-72. ACM, 2009.

[9] David B. Thomas, Wayne Luk *FPGA-Optimised Uniform Random Number Generators using LUTs and Shift Registers*. 2010 International Conference on Field Programmable Logic and Applications. IEEE.

[10] Jansson, Birger. *Random number generators (1966).*

[11] Knuth, Donald E. *Seminumerical algorithms*. (2007).

[12] David B. Thomas, Wayne Luk. *High Quality Uniform Random Number Generation Using LUT Optimised State-transition Matrices*. Journal of VLSI Signal Processing 47, 7792, 2007.

[13] Alin Suciu, Radu Alexandru Toma, Kinga Marton. *Parallel Implementation of the TestU01 Statistical Test Suite* Intelligent Computer Communication and Processing (ICCP), 2012 IEEE International Conference on (pp. 317-322). IEEE.

[14] Abdel-Rehim, Wael MF, Ismail A. Ismail, and Ehab Morsy. *Testing randomness: Implementing poker approaches with hands of four numbers*. International Journal of Computer Science Issues 9, no. 4 (2012).

7 Appendices

7.1 Appendix 1 - Gantt Chart

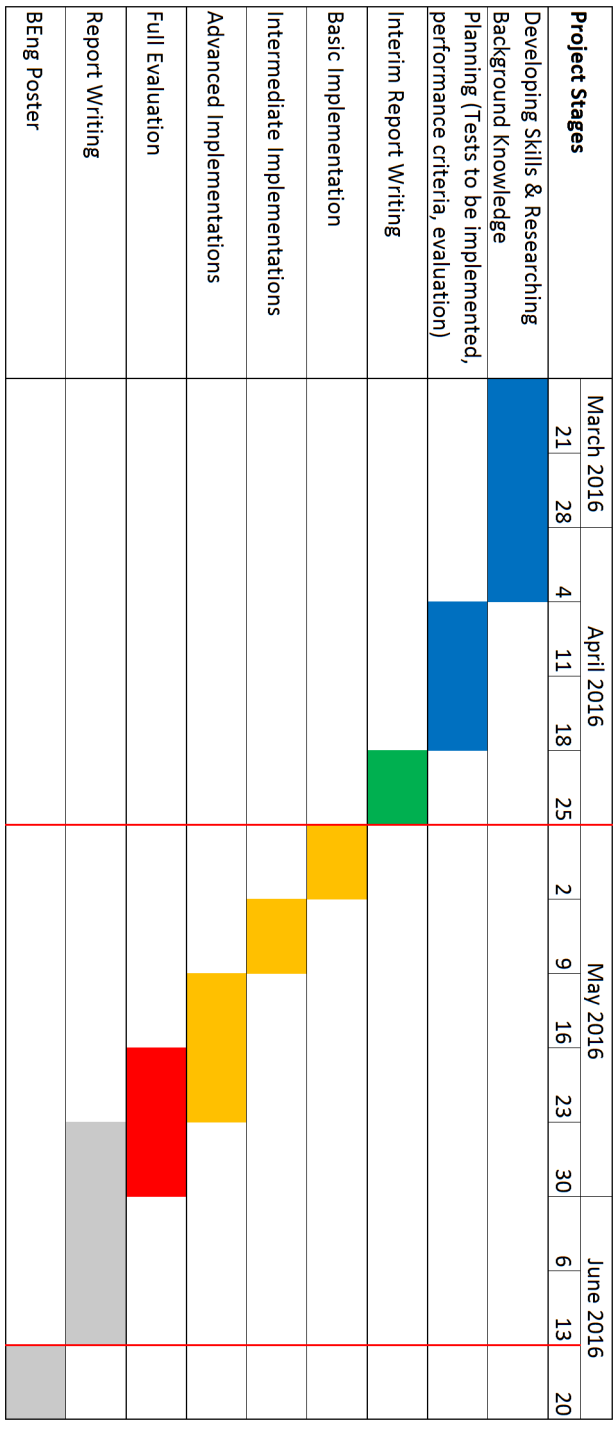


Figure 2: Gantt Chart with weekly breakdown of project stages