A Comparison of Physical Random Number Generators

Winner, Science

Author: Logan Francis

Editor's Note: The limitations of SMU's content management system make it impossible to properly represent some mathematical symbols and equations used in this paper. To read it as the author intended, please contact The Writing Centre for a copy of the print edition of Afficio Vol. 2.

Abstract

Good random number generators (RNGs) are required for many applications in science and industry. Random numbers can be created in two ways: with a computer algorithm known as a pseudo-random number generator (PRNG), or by measuring physical phenomena which behave randomly, such as quantum mechanical or chaotic systems. However, PRNGs are deterministic in nature and cannot produce truly random output, while physical RNGs can. Three physical RNGs were constructed: a Chua circuit, an electrical circuit which exhibits chaos; an avalanche circuit, which produces a noisy electrical signal; and a radioactive decay counter. Each RNG produced output in the form of ASCII files containing 0s and 1s. The randomness of the data was assessed using the open source statistical test suite **rngtest.**

Introduction

Random number generators (RNGs) are ubiquitous in science and mathematics, used in applications ranging from physics simulations to cryptography to bioinformatics. For most applications, a *pseudo-random number generator* (PRNG) is employed. A PRNG uses an algorithm to produce a sequence of numbers which appear to be random from input *seed* numbers, which are used to calculate the first terms in the sequence [1]. While PRNGs are very easy to implement in a piece of software, they have several major drawbacks. The first problem is if the seed numbers are known, the entire sequence of random numbers can be replicated by supplying them to the PRNG. This is a problem in cryptography, where knowing these seed numbers could allow an interloper to decode an encrypted file. Second, only a finite sequence of random numbers is produced by a PRNG, the length of which is called the *period. [2].* After one period of random numbers has been produced, a PRNG typically continues from the beginning of the sequence. This could be a problem for a computer simulation using Monte Carlo methods, which use random numbers to perform tasks such as integration of functions. Many PRNGs today have longer periods to avoid this problem, such as the Mersenne Twister PRNG, which has an extremely long period of 2^{19937} -1 [3]. Worst of all, PRNG output does not necessarily behave very randomly at all. The RANDU PRNG developed in the early 1960s was notoriously terrible: if triplets of consecutive numbers produced by RANDU are plotted in 3-D, the output forms the pattern of the planes seen in Figure 1. Qualitatively, one would expect a plot of the same nature produced by a good RNG to resemble white noise and feature no obvious patterns.

FIG. 1. A 3-D plot of triplets of sequential numbers in the range [0,1] produced by the flawed RANDU PRNG. [1]

When a PRNG is inadequate for the above reasons, a hardware or *physical* RNG is used instead. Physical RNGs make measurements on a physical process in order to produce truly random numbers. While they do not have a period or require seed numbers, physical RNGs are generally slower than PRNGs because they are limited by how fast measurements can be made and processed, whereas a PRNG is limited only by the speed of the computer running it.

The randomness of a physical RNG is guaranteed by the physics of the system on which it performs measurements. In most cases, the randomness is ultimately caused by making measurements on a quantum mechanical system. For example, the position of an electron is described by quantum mechanics using a wave function that represents the probability of an electron being in a certain position in space [4, p.14]. The position of the electron is unknown until the measurement is made and cannot be predicted (i.e., it is random); only the distribution of the result of many measurements can be known.

Another possible source of randomness is to measure a chaotic system. While a chaotic system behaves in a deterministic way, *sensitivity of initial conditions* limits how the behaviour of the system can be predicted. Only if the initial conditions of the system are known *exactly* can the evolution of the system be predicted for all time. Since this is impossible, measurements made on a chaotic system whose evolution cannot be predicted can be used to produce random numbers.

The randomness of a RNG can be assessed using statistical tests to examine whether or not the output behaves randomly. For example, a good RNG should produce 0s and 1s with equal probability, and over a long period of time, output a roughly equal amount of each.

Experiment

A set of three physical RNGs each based on a different physical system were constructed to assess the quality of their output.

Avalanche Noise RNG

Avalanche breakdown is a phenomenon that creates noise in electrical circuits known as *avalanche noise.*Avalanche breakdown occurs in a semiconductor with a large voltage gradient. There is a chance that an electron in the semiconductor may break free from an atom, and begin moving towards the positive applied voltage in the circuit, while the positive electron hole left behind will move towards the negative terminal. If the now free electron has a large enough kinetic energy, it may knock additional electrons free, creating new pairs of electrons and electron holes. This can cause a chain reaction, resulting in a current fraught with avalanche noise suddenly flowing through the semiconductor [5]. Since the probability of electrons around an atom in the semiconductor being removed in a collision with a free electron is quantum mechanical in nature, this avalanche noise could potentially be used as a source of random numbers.

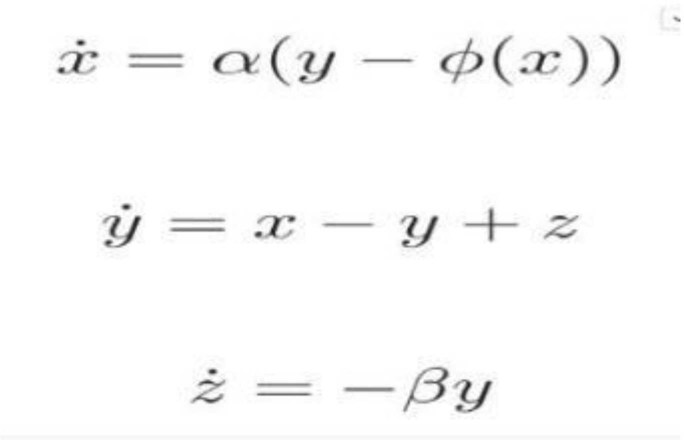
An avalanche noise RNG was created on a bread board and connected to an Arduino Mega 2560 microcontroller, using the design by Rob Seward in Figure 2 [6]. The Arduino board was in turn connected to a PC running Windows 7, which was used for recording data and exporting software to the Arduino board. The circuit worked by producing avalanche noise at the junction between the two transistors [6], then amplifying the signal using the third transistor before finally sending it to the analog input pin of the Arduino board. The Arduino board ran a piece of software which wrote a 1 or 0 to a output console in the Arduino development software on the PC when the voltage was above or below the median voltage in the circuit respectively, where the median was determined by an automatic 10 second calibration test [6] at the beginning of each run.

A total of eight runs of data were collected from the circuit, each recorded over a period of roughly a half hour. Data from the console was copied to a text file at the end of each run. The length of a data set was limited by the stability of the software used to interact with the Arduino board; after about 40 minutes, the serial port monitor used to record data would crash, and any data stored in it would be lost. However, the data sets were still much larger than the lower limit of 20,000 required by the program used for analysis.

FIG. 2. Circuit Design for the avalanche RNG. The boxes on the side are the input pins of the Arduino board.

Chua Circuit RNG

A *Chua circuit* is a simple electric circuit which exhibits chaotic effects. It models the set of non-linear differential equations



$$\dot{x} = \alpha(y - \phi(x))$$

$$\dot{y} = x - y + z$$

$$\dot{z} = -\beta y$$

where *(x)* is a nonlinear function. Plotting the z state variable versus the x state variable on an oscilloscope produces the pattern shown in Figure 3, a strange attractor. A strange attractor is a region of phase space (the space formed by the variables in the system of equations) that the chaotic system tends to evolve to.

The Chua circuit was constructed on a breadboard using a design by Giorgio Vazzana [7], and calibrated using a potentiometer in the circuit until the strange attractor shown in Figure 3 appeared on the Tektronix TDS 1001B oscilloscope connected to the nodes in the circuit representing the z and x state variables.

Two methods were used to produce random output. An oscilloscope was first used to sample the waveform of the z state variable with a 200ms sample interval. The output spreadsheet was saved to a flash drive connected to the oscilloscope once the maximum amount of data (2200 sample points) was recorded. A limit was placed on the size of the data sets since once the oscilloscope recorded more than 2200 points, it began to write over the old points. A total of 18 data sets of 2200 points each were recorded this way. The Libre Office Calc spreadsheet program was used to copy the voltage column of the spreadsheet to a text file.

This system was later upgraded to perform the sampling by connecting the same points in the circuit to a FLUKE-8845a multimeter, in turn connected to a Windows 7 PC running a labview program which recorded the signal.

To transform the output to a binary sequence, the Libre Office Calc spreadsheet program was used to copy the voltage column to a text file. This data was then analyzed with a simple **FORTRAN** program which wrote a 1, 0, or nothing to disc for positive, negative, and zero voltages respectively.

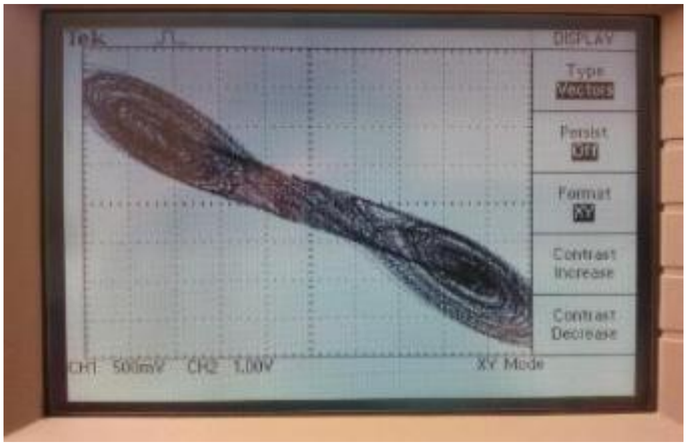


FIG. 3. An oscilloscope plot of the z state variable versus the x state variable of the Chua circuit used for the Chua circuit RNG. Strange attractors are located in the centre of the disc-shaped regions.

Radioactive Decay RNG

The process of radioactive decay is described by quantum mechanics. Taking alpha decay as an example, the alpha particle is held within the nucleus by a region of lower nuclear binding potential energy [4, p.334]. For the alpha particle to be ejected from the nucleus, it must tunnel through a region of higher potential energy, where it will then be repelled from the nucleus by the Coulomb force, since both an alpha particle and nucleus are positively charged. The alpha particle has a probability of tunneling through the barrier, which can only be used to tell how likely the particle is to escape in a given time interval, but not at what time it actually *will* escape.

The experimental setup for the radioactive decay RNG consisted of a Geiger Counter connected to a Data Acquisition Unit (DAQ), which was in turn connected to the same PC used for the avalanche RNG, running a labview program to collect the data. Both background radiation and a Caesium-137 source were used with the Geiger counter to provide radioactive decay. The labview program received input from the DAQ in the form of a count of the number of radioactive decays. A timer was constructed in the labview program which recorded the time of each decay and wrote it to a text file as it occurred. Six sets of data were collected, 3 of which used background radiation, and 3 of which used a radioactive source.

A program was written in **FORTRAN** to convert the decay times into binary data using the algorithm for the Hotbits [8] radioactive decay RNG as a basis:

1. From the first 4 detected decays, generate two time intervals T_1 and T_2 by subtracting the time of the second decay from the first, and the time of the fourth decay from the third.
2. Compare the lengths of T_1 and T_2 as shown in figure 4

3. If $T_2 > T_1$, write a 1 to disc. If $T_2 < T_1$, output a 0 to disc. If $T_2 = T_1$, output nothing and proceed to the next step.
4. Reverse the direction of the comparison between T_1 and T_2. This step prevents output from being biased towards the production of zeros or ones.
5. Select the next 4 decays and repeat from step 1 until end of file.

The output of this **FORTRAN** program was saved as ASCII text files.

FIG. 4. Time intervals between decays used to produce random output for the radioactive decay RNG [8].

Bias Removal

Many physical RNGs exhibit a bias towards producing more 1s or 0s in their output stream. This is caused by the difficulty of balancing the physical phenomenon measured such that the probability of emitting a 0 or 1 is equal. For example, the Chua circuit may be biased because of the signal spending more time around one strange attractor than another.

Fortunately, if the physical RNG otherwise behaves randomly, this bias can be easily corrected. Von Neumann bias removal removes all bias from a random sequence using the following algorithm [9]:

1. Choose the next pair of successive, non-overlapping
2. If the bits in the pair are equal, discard them, and go to step 1.
3. If the bits are not equal, output the first bit in the
4. Go to step 1.

The downside to the Von Neumann algorithm is that it reduces the size of the data set; after applying it, the resulting data set is at most 25% of the size of the original set.

The bias removal was implemented as described using a simple **FORTRAN** program. While it was used to remove bias from the Chua and avalanche RNGs, it was not needed for the radioactive decay RNG because its algorithm incorporates bias removal by virtue of the comparison reversal in its fourth step.

Random Output Analysis

All of the aforementioned physical RNGs produce output in the form of ASCII files containing sequences of 0s and 1s. The software **rngtest** used to perform further analysis of this data required input in a binary format, the conversion of which was done by the c program asci_to_bin [7] written by Giorgio

Vazzana. **rngtest** implements five statistical tests required by the American National Institute of Standards and Technology (NIST) for cryptographic security standards. It performs these five tests on blocks of 20,000 [10] bits at a time and gives a pass or fail rating to each block for each test [11].

The continuous runs test is a basic test that is run on every set of blocks passed to **rngtest.** It is failed if any block in the sequence is identical to another, and is intended to catch a total failure of the RNG.

The long runs test is passed if there are no runs of 1s or 0s of length 34 or greater, which are extremely unlikely ($p = 1/2^{34}$) and probably indicate a malfunction of the RNG.

The monobit test checks for bias in the output of the RNG by comparing the total number of 1s and 0s in the sequence. A RNG should have equal probability of producing a 1 or 0, otherwise the RNG is biased towards a given output. The monobit test is considered passed if the number of is 1s in the range 9654 to 10346.

The poker test simulates the drawing of idealized poker hands from a deck of cards. This is done by dividing the block of 20,000 points into 5000 4 bit hands, and counting the number of each of the 16 possible hands that occur. A chi-squared test is then performed on the observed distribution of hands, which compares the observed distribution of hands to the predicted one and checks if the probability of this distribution occurring is reasonable [12, p. 39]. The test takes the form

$$\chi = \left(\frac{16}{5000}\right) * \left(\sum_{i=0}^{15} f(i)^2\right) - 5000$$

where *f(i)* contains the number of each combination of hands. The test is considered passed if 1.03 < < 57.4.

The runs test counts the number of sequences of consecutive 1s and 0s and checks if the count of runs of a given length is normally distributed. For example, the sequence 10001110000 contains two runs of length 3, and a single run of length 4. For the block of 20,000 bits used by **rngtest,** the counts of runs of both 1s and 0s must fall within the range specified in Table I to pass this test.

Results and Discussion

Every RNG tested passed the continuous runs and long runs test, indicating that no generator was failing by producing only a stream of 1s or 0s. The results of the monobit, poker, and runs tests are summarized in Table II**.** Much more data was collected from the avalanche RNG than the Chua and radioactive decay RNGs because of its relatively high output rate; in a typical half hour run, the avalanche RNG would produce 500,000 bits of output, the Chua RNG, 5000, and the radioactive decay RNG, 10,000.

The avalanche RNG with bias removal implemented passes every test of randomness. The results of the tests on 25 blocks of data from the avalanche RNG without bias removal are shown in Table III**.** Here, the avalanche RNG fails at least one of the monobit, poker, or runs tests in 13 of the 25 blocks. Bias removal is necessary for the avalanche RNG because the output of a 1 or 0 depends on comparison of the present voltage with the median voltage calculated during a calibration step before taking data. The median voltage could drift while taking data, resulting in biased output from the generator. A better procedure for reducing bias could be to update the median voltage used for comparison in another calibration step after a certain interval of time or amount of output.

TABLE 1. Criteria for passing the runs test in **rngtest** [11].

Length of Run	Passing Range of Number of Runs
1	2,267 -2,733
2	1,079 – 1,421
3	502 – 748
4	223 – 402
5	90 – 223
6	90 – 223

The two blocks of data analysed for the Chua circuit without bias removal failed the monobit, poker, and runs tests. With bias removal, data sets from the Chua circuit no longer failed the monobit test, but continued to fail the poker and runs tests, indicating that the output was not truly random rather than merely biased. Relatively little data was available for testing from the Chua circuit because of the sampling interval of 200ms used. This interval translates to an output rate of 18,000 bits per hour before bias removal, which reduces the maximum output rate to only 4,500 bits per hour. To determine what went wrong with the Chua circuit, it would be useful to examine the frequency spectrum of the z state variable. A randomly fluctuating signal would be expected to contain an even mix of all frequencies. Any periodic variation in the signal would show result in a spike at a corresponding point in the frequency spectrum. It would also be helpful to see if changing the sampling interval has any effect on the randomness of the output. Decreasing the sampling interval would be useful for increasing the output rate of the Chua circuit once it is known to produce truly random output.

The data set for the radioactive decay RNG using a Cs-137 source was produced by merging three separate runs to meet **rngtest's** minimum requirement of 20,000 bits per block. All 5 blocks in the set passed all tests of randomness. Not enough data was collected from runs with only background radiation to meet **rngtest's** minimum requirement because of the very low rate of detection. The output rate of the RNG could be improved by using a more radioactive (but potentially more dangerous) source or a more sensitive Geiger counter.

Conclusions

Of the three methods of producing random numbers tested, the avalanche noise circuit with bias removal reliably produces random numbers at the highest output rate. The output from the Chua circuit without bias removal failed to pass the monobit, poker, and runs tests. The addition of bias removal caused the same data sets to pass the monobit, but still fail the poker and runs tests. Since it is unknown exactly why the Chua circuit failed to produce random output and little data was available for analysis, the Chua circuit cannot be said to be unsuitable for producing random numbers in general without further testing. The data from the radioactive decay RNG passed all tests of randomness, but produced output approximately 50 times slower than the avalanche RNG. However, the need for a radioactive source and Geiger counter makes it less practical to implement than the avalanche and Chua circuits, which require only simple electrical components.

TABLE II. Results of the monobit, poker, and runs tests performed by **rngtest**.

Data Set	Set Length (bits)	Blocks	Number of Test Failures			Successful Blocks	Failed Blocks
			Monobit	Poker	Runs		
Avalanche RNG run 1	221152	11	0	0	0	11	0
Avalanche RNG run 2	127800	6	0	0	0	6	0
Avalanche RNG run 3	157528	7	0	0	0	7	0
Avalanche RNG run 4	206976	10	0	0	0	10	0
Avalanche RNG run 5	120032	6	0	0	0	6	0
Avalanche RNG run 6	95752	4	0	0	0	4	0

Avalanche RNG run 7	159032	7	0	0	0	7	0
Avalanche RNG run 8	155552	7	0	0	0	7	0
Avalanche RNG run 9	184544	9	0	0	0	9	0
Avalanche RNG run 10	246296	12	0	0	0	12	0
Chua RNG run 1	41824	0	0	2	2	0	2
Rad. Decay RNG run 1	105024	5	0	0	0	5	0

TABLE III. Avalanche RNG **rngtest** results with no Von Neumann bias removal.

	Set Length		Number of Test			Successful	Failed
Data Set	(bits)	Blocks	Failures			Blocks	Blocks
			Monobit	Poker	Runs		
Avalanche RNG run 2	510856	25	13	10	9	12	13

References

[1] Evans, "Simulating random numbers with a computer," http://hep.physics.indiana.edu/-hgevans/p410-p609/materia1/04_rand/prng_types.html/ (2010).

[2] Haar, "Introduction to randomness and random numbers," http://www.random.org/randomness/.

[3] N. Makoto Matsumoto, "What is mersenne twister (mt)?" http://wwri.math.sci.hiroshima-u.ac.jp/m-mat/MT/ewhat-is-mt.html/ (2013).

[4] J. Griffiths, *Introduction to Quantum Mechanics Second Edition* (Pearson, 2005).

[5] Poole, "Avalanche noise - the basics of avalanche noise - how it is created, how it can be used or removed," http://www.radio-electronics.com/info/rf-technology-design/noise/avalanche-noise-basics.php.

[6] Seward, "Make your own true random number generator 2," http://robseward.com/misc/RNG2/.

[7] Vazzana, "Random sequence generator based on chua circuit," http://holdenc.altervista.org/chua/ (2012).

[8] Walker, "How hotbits works," http://wmi.fourmilab.ch/hotbits/how3.html.

[9] V. Neumann, National Bureau of Standards Applied Math Series 12, 36 (1951).

[10] de Moraes Holschuh, "rngtest man page," http://www.linuxcommand.org/man_pages/rngtest1.html (2004).