

Random Number Generators*

Charles B. Cameron

November 24, 2008

1 Introduction

Why would there ever be any need to generate random numbers? There are numerous applications. Here are a few of them:

- To generate random keys for use in encryption.
- To generate lottery numbers.
- For performing experiments in extra-sensory perception (ESP).
- For performing Monte Carlo simulations, in which parameters of a system may be subject to random variability. By doing enough simulations, you can get an idea of the effect of these variations on the overall system.
- For testing error-correction schemes. Using random numbers makes it possible to introduce arbitrarily selected bit-error rates.
- For testing communications demodulation schemes, for the very same reason.
- For use in games that need a random component. For example, games with an unpredictable starting state or with unpredictable playing parameters, such as difficulty.

Broadly speaking, there are two classes of hardware to generate random numbers:

- truly random number generators and
- pseudorandom number generators.

Most of the truly random number generators rely on the randomness associated with noise in resistors, semiconductors, or radioactive sources. Pseudorandom number generators, on the other hand, attempt to simulate randomness through mathematical techniques.

*Course notes for EE361 Microprocessor-based Digital Design

2 Truly Random Number Generators

2.1 Johnson Noise

The Johnson noise voltage generated by thermal activity in a resistor has an extremely broad, flat spectrum. Such noise includes all wavelengths within this spectral range and they all have equal power or intensity. It is the presence of more-or-less equal amounts of energy at all visible wavelengths that makes white light look white to us, so Johnson noise is often referred to as *white* noise, generated by thermal activity. The probability distribution of the voltages is Gaussian, meaning the voltages are normally distributed. A Gaussian probability distribution function has the form

$$f(\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where

$$\begin{aligned}\mu &= \text{the mean value of } f(x) \text{ and} \\ \sigma &= \text{the standard deviation of } f(x).\end{aligned}$$

The mean value μ of $f(x)$ is its average value and the standard deviation σ is a measure of the extent to which $f(x)$ deviates from the mean. The probability that $f(x)$ is within the range $\mu \pm \sigma$ is over 68%; the probability that $f(x)$ is within the range $\mu \pm 2\sigma$ is over 95%; and the probability that $f(x)$ is within the range $\mu \pm 3\sigma$ is over 99%.

Johnson¹ noise voltage has a mean value $\mu = 0$ V and a root-mean-square voltage given by

$$v_{\text{RMS,JOHNSON}} = \sqrt{4kTRB}$$

where

$$\begin{aligned}k &= \text{Boltzmann's constant} = 1.38 \times 10^{-23} \text{ J/K,} \\ T &= \text{absolute temperature,} \\ R &= \text{the resistance of the resistor, and} \\ B &= \text{the bandwidth of a (hypothetical) perfect, noiseless, bandpass} \\ &\quad \text{filter whose input is driven by the resistor.}\end{aligned}$$

This voltage $v_{\text{RMS,JOHNSON}}$ corresponds to the standard deviation σ of a voltage probability distribution. Because the voltage is random, we can obtain a random number with an underlying Gaussian probability distribution from this voltage by, say, passing it through an analog-to-digital converter. Typically it would be

¹Johnson discovered in 1928 that all conductors have a non-periodic voltage associated with them, a voltage generated by thermal effects. Nyquist found the mathematical formula for Johnson noise, reasoning from thermodynamic principles.[1]

necessary to amplify and possibly shift the random voltages in order to match the input range of the A/D converter. In the case of an A/D converter with strictly positive input voltages, say 0 V to 5 V, the digital output of the A/D converter would be non-zero but it might not be at the mid-point of the output range. The average would depend on the average voltage at the input to the A/D converter.

In practice, control over the standard deviation σ depends either on controlling the resistance R or the temperature T .

2.2 Shot Noise

Another means of generating truly random numbers relies on noise generated in semiconductors, such as diodes. This noise is referred to as *shot* noise, possibly because it sounds a bit like shotgun pellets falling onto a surface. Like Johnson noise, the amplitude of shot noise has a Gaussian distribution with zero mean. However, this is current noise, not voltage noise. A steady current passing through the diode is subject to random fluctuations about its average value. The root-mean-square amplitude of these fluctuations is

$$i_{\text{RMS,SHOT}} = \sqrt{2qI_{\text{DC}}B}$$

where

q = an elementary charge = 1.602×10^{-19} C,

I_{DC} = the (nominal) direct current through the diode, and

B = the bandwidth of a (hypothetical) perfect, noiseless, bandpass filter whose input is driven by the diode.

Whereas Johnson noise $v_{\text{RMS,JOHNSON}}$ relates to random voltage fluctuations, the shot noise $i_{\text{RMS,SHOT}}$ corresponds to the standard deviation σ of a current probability distribution. Because the current fluctuation is random, we can obtain a random number with an underlying Gaussian probability distribution from this current by, say, sending it through a resistor and passing the resultant voltage through an analog-to-digital converter. Because Johnson noise represents a zero-mean fluctuation around a non-zero DC current, the average DC current will generate an average DC voltage, giving the A/D converter's input and output a non-zero average.

In practice, control over the standard deviation σ depends on controlling the direct current I_{DC} . This tends to be much easier to control than changing either temperature T or resistance R in the case of Johnson noise and so is preferable in that respect.

2.3 Generating a Uniformly Distributed Random Number

The schemes discussed above convert physical electrical noise into digital numbers with a Gaussian distribution around a mean. One way of varying the

scheme is to use an analog comparator to generate either a 1-bit or a 0-bit, depending on whether or not the random voltage crosses its switching threshold or not. If the threshold is set to correspond to the average value of the input voltage, then a 1-bit and a 0-bit will be generated with equal probability. By collecting a sequence of n of these values, a truly random number of n bits can be created.

There is a catch, however: voltages do not change appreciably in zero time. Therefore, successive voltages are correlated with each other. However, waiting long enough between samples is sufficient to let the correlations die away, effectively making the samples independent of one another. How long is long enough? Sampling intervals Δt should be much greater than the reciprocal of the bandwidth, that is, we require

$$\Delta t \gg \frac{1}{B}.$$

For example, if we use a bandpass filter with bandwidth $B = 1$ MHz, we should choose

$$\Delta t \gg \frac{1}{1 \text{ MHz}} = 1 \mu\text{s}.$$

A sampling interval of, say, $10 \mu\text{s}$ to $100 \mu\text{s}$ would suffice in this case. Reducing the bandwidth to $B = 1$ kHz, on the other hand, would require lengthening our sampling interval to 10 ms to 100 ms.

What happens if the random noise source does not have an average value that is halfway between the lower voltage reference V_{REF}^- and the upper voltage reference V_{REF}^+ of the A/D converter? The result will be a biased random number generator. One way to eliminate the bias would be to take successive input bits two at a time, as described in RFC1750[2]. Discard pairs of identical bits, taking only pairs that consist in bits 01 or 10. If the bit pair is 01, output a 0; if it is 10, output a 1. Provided the random bits are truly uncorrelated, this will completely eliminate any bias. Since some bit pairs will likely be discarded, it will take an indeterminate time to collect n random bits. As RFC1750 shows, the expected time to collect n bits depends on the degree to which the probability of input bits being 0 or 1 differs from 0.5. If a zero-bit occurs with probability $p_0 = 0.5 + e$ and a one-bit occurs with probability $p_1 = 0.5 - e$, then on average you will need approximately $N = n / (0.25 - e^2)$ input bits ($n/2$ input bit pairs) before you will have collected n random output bits.

The table below shows a few combinations of these values:

n	e	p_0	p_1	N
100	0.001	0.501	0.499	400.0
	0.01	0.510	0.490	400.2
	0.1	0.600	0.400	416.7
	0.2	0.700	0.300	476.2
1 000	0.001	0.501	0.499	4000.0
	0.01	0.510	0.490	4 001.6
	0.1	0.600	0.400	4 166.7
	0.2	0.700	0.300	4 761.9
10 000	0.001	0.501	0.499	40 000.2
	0.01	0.510	0.490	40 016.0
	0.1	0.600	0.400	41 666.7
	0.2	0.700	0.300	47 619.0

A casual inspection of the values for N —or, for that matter, the formula for N —shows that we need to collect roughly $4n$ bits, on average, to obtain n bits with no bias, provided the source of input bits is not too badly biased.

3 Pseudorandom Number Generators

There are several pseudorandom number generators in widespread use. Most of these try to generate bit sequences having equal probability for 1 and 0 bits.

3.1 Linear Congruential Generators

An m -bit pseudorandom number V_j is calculated from a previous pseudorandom number V_{j-1} by the recurrence relation

$$V_j = AV_{j-1} + B \pmod{m}$$

where A and B are constants particular to a specific generator. The sequence of random numbers repeats with a period of at most m , but often it is less than this, depending on the choices for A and B .

There are other defects to the scheme. For example, successive numbers are correlated, making it possible to infer a new one from a previous one. Also, low-order bits in the numbers tend to repeat with a period much shorter than the overall period.

3.2 Primitive Polynomials modulo m

Another commonly used scheme depends on the properties of polynomial division. A polynomial with binary coefficients is of the form

$$p(x) = \sum_{i=0}^m g_i x^i$$

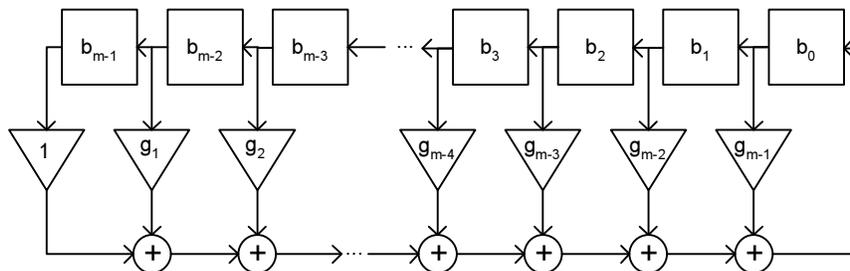


Figure 1: A generalized Fibonacci implementation of the linear feedback shift register.

where the coefficients g_i can take on either the value 0 or the value 1.

We can implement a pseudorandom number generator based on such a polynomial as a linear feedback shift register with m bits $b_{m-1}, b_{m-2}, \dots, b_2, b_1, b_0$.

There are two widely used implementations.[3]

Fibonacci Implementation

$$b_j \leftarrow \begin{cases} (b_0 \cdot g_{m-1}) \oplus (b_1 \cdot g_{m-2}) \oplus \dots \oplus (b_{m-2} \cdot g_1) \oplus b_{m-1} & \text{if } j = 0 \\ b_{j-1} & \text{if } j \neq 0. \end{cases}$$

Galois Implementation

$$b_j \leftarrow \begin{cases} b_{m-1} & \text{if } j = 0 \\ b_{j-1} \oplus b_{m-1} & \text{if } j \neq 0 \text{ and } g_j = 1 \\ b_{j-1} & \text{if } j \neq 0 \text{ and } g_j = 0 \end{cases}$$

A generalized schematic of the Fibonacci implementation can be seen in Figure 1. When the polynomial weights g_i are 0, they can be implemented as open circuits. When they are 1, they can be implemented as wires. The \oplus symbols represent an exclusive-OR function, equivalent to ordinary binary addition without a carry. Note that weight g_{m-1} is closest to the shift register's input.

For example, the particular polynomial $p(x) = x^5 + x^4 + x^2 + 1$ can be implemented as shown in Figure 2. (Note: this is not a good polynomial to use since it will not generate all $2^5 - 1 = 31$ non-zero values. It is used only for the purpose of illustration.) The highest exponent indicates the number of bits in the shift register. The lowest-order term always is 1. As for the other exponents, a term in x^i is present if its corresponding weight $g_i = 1$ and it is missing if $g_i = 0$.

An alternative to the Fibonacci implementation is the Galois implementation. A generalized version of it is shown in Figure 3. The gains g_i are exactly the same as those of the Fibonacci implementation. Note, however, that the weights in the Galois implementation appear in the reverse order from that of

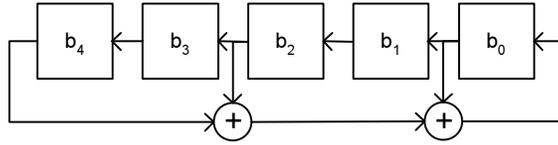


Figure 2: A Fibonacci implementation for the polynomial $p(x) = x^5 + x^4 + x^2 + 1$

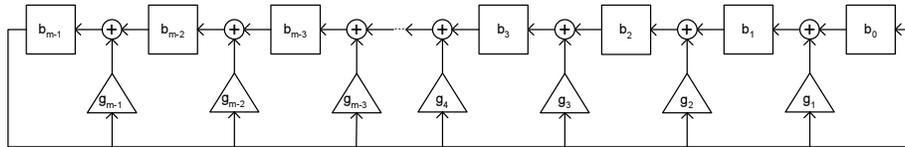


Figure 3: A generalized Galois implementation of the linear feedback shift register.

the Fibonacci implementation. Weight g_{m-1} now is closest to the output of the shift register, not the input, as with the Fibonacci implementation. Although both implementations produce the same sequence of pseudorandom codes, it takes different starting values to get them to do so.

The same polynomial, $p(x) = x^5 + x^4 + x^2 + 1$, whose Fibonacci implementation was shown in Figure 2 has the Galois implementation of Figure 4.

The starting value for both the Fibonacci implementation and the Galois implementation is very important. In particular, it must not be all zeros, for if it is, then the contents of the register will be zero always. The initial value is called the *seed* value.

Although the sequences may appear to be random and may pass a number of tests showing an equal likelihood of any particular non-zero value ever being generated, the sequences are in fact completely deterministic.

This pseudorandom number generation scheme has two main flaws:

- If the polynomial is not chosen with care, the system may operate in a sub-cycle, meaning that only a subset of all possible register values will occur. Mathematical analysis makes it possible to pick polynomials that do not suffer this defect, polynomials that generate all $2^m - 1$ non-zero values.
- It is not a good idea to use them for encryption because although they

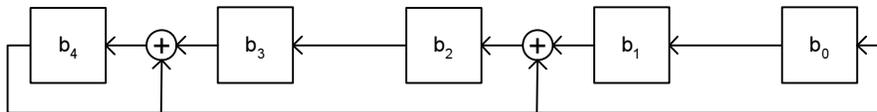


Figure 4: A Galois implementation for the polynomial $p(x) = x^5 + x^4 + x^2 + 1$

Number of Bits	Number of Taps	Tap Points
32	4	32, 31, 30, 10
	6	32, 31, 30, 29, 26, 16
	16	32, 31, 29, 26, 24, 23, 21, 18, 16, 15, 13, 10, 8, 7, 5, 1
30	4	30, 28, 27, 6
	6	30, 29, 28, 26, 24, 9
	16	30, 29, 26, 25, 23, 20, 19, 16, 14, 13, 10, 9, 8, 7, 4, 3
16	4	16, 14, 9, 4
	6	16, 15, 14, 11, 10, 6
	14	16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 3, 2

Table 1: Some useful m -sequences of various lengths.[3] The tap points indicate the terms in the polynomial that have non-zero coefficients. For example, the four tap points 32, 31, 30, and 10 correspond to the order-32 polynomial $p(x) = x^{31} + x^{30} + x^{10} + 1$ and the six tap points 16, 15, 14, 11, 10, and 6 correspond to the order-16 polynomial $p(x) = x^{15} + x^{14} + x^{11} + x^{10} + x^6 + 1$.

can be made to provide sequences that appear to be random, the ciphers based directly on them can be broken fairly easily.[4]

A linear feedback shift register cannot contain all zeros without ceasing to function. This exception aside, however, when the generating polynomial is a so-called m -sequence, all non-zero numbers occur with equal probability. These m -sequence polynomials have numerous useful properties and are widely tabulated. Table 1 lists a few of them for various useful lengths. The tap points indicate the terms in the polynomial that have non-zero coefficients. For example, tap points 32, 31, 30, and 10 correspond to the order-32 polynomial $p(x) = x^{31} + x^{30} + x^{10} + 1$. Note that even though the list of tap points does not mention 0 explicitly, there always is a feedback from the most significant to the least significant bit.

4 Generating Bit Errors at a Specified Rate

Once we have a random number, whether generated by a truly random number generator or by a pseudorandom number generator, it is easy to construct a circuit to create any desired rate of bit errors, provided you can generate the random numbers rapidly enough. Suppose we are using m -bit random numbers, for example. To get a bit error rate $\rho \in [0, 1]$, compute the value

$$q = \rho(2^m - 1).$$

Whenever the random number $x \leq q$, complement the transmitted bit. This will happen just often enough to keep the expected bit error rate equal to ρ .

References

- [1] H. W. Ott, *Noise Reduction Techniques in Electronic Systems*. New York: John Wiley & Sons, 1976.
- [2] D. Eastlake, 3rd, S. Crocker, and J. Schiller, “RFC 1750: Randomness recommendations for security,” url = “<http://rfc.net/rfc1750.html>”, Dec. 1994, accessed 8 April, 2005.
- [3] New Wave Instruments, “Linear feedback shift registers,” url = “http://www.newwaveinstruments.com/resources/articles/m_sequence_linear_feedback_shift_register_lfsr.htm”, accessed 8 April, 2005.
- [4] E. Zenner, “Cryptanalysis of LFSR-based pseudorandom generators,” Technical Report TR-04-004, Department for Mathematics and Computer Science, University of Mannheim, Tech. Rep., 2004, available online at http://134.155.36.45/madoc/frontdoor.php?source_opus=727.
- [5] G. R. Cooper and C. D. McGillem, *Probabilistic Methods of Signal and System Analysis*, 2nd ed. New York: CBS College Publishing, 1986.
- [6] P. Horowitz and W. Hill, *The Art of Electronics*, 2nd ed. New York: Cambridge University Press, 1989.