Comments Received on DRAFT ANS X9.82
Random Number Generation

From: "Steve Crocker" <steve@stevecrocker.com>
Subject: RFC 1750 re Random Number Workshop
Date: Sat, 26 Jun 2004 20:28:43 -0400

Your message about the upcoming workshop on generating random numbers just came to my attention. This is an important subject and one that has not had enough attention. Many of the people who build commercial security systems have too little understanding of the entropy issues and have occasionally released software which has been broken with surprising ease. For example, Netscape's browser used very poorly generated random numbers in the mid-90s, which two Berkeley grad students discovered and broke, causing Wells Fargo to immediately suspend their online banking operation.

The crux of the problem is the lack of official hardware entropy sources on standard computers. There is no instruction in the standard instruction set that says "generate an unreproducible result." In fact, the philosophy is exactly the opposite: running the same code is supposed to generate the same result. Of course, as we know, modern computers do have sources of non-determinism at the edges of their design -- mouse movement, disk movement, network traffic, etc. The trick is how and when to use these sources. Code written to get random bits from disk movement won't do very well on a diskless machine. Code written to sample network traffic won't do very well on a stand-alone machine. Etc.

When I was IETF Security Area Director in the early 1990s, we initiated work to document and educate software developers working in this area. The result was RFC 1750, Randomness Recommendations for Security. It was issued in December 1994 as a Best Current Practice and has remained an important source of guidance for developers. Although I am a co-author on this, Donald Eastlake has done the bulk of the work with Jeff Schiller pitching in on critical points.

This RFC is undergoing revision to bring it up to date. I'm attaching the Internet-Draft that's been submitted for publication to replace RFC 1750.

From its abstract:

> This document points out many pitfalls in using traditional pseudo-random number generation techniques for choosing such quantities. It recommends the use of truly random hardware techniques and shows that the existing hardware on many systems can be used for this purpose. It provides suggestions to ameliorate the problem when a hardware solution is not available. And it gives examples of how large such quantities need to be for some applications.

The main emphasis of this document is in the last two sentences, viz how to extract entropy when no official hardware source is available.

I note that neither the ANSI draft standard nor your announcement mentions RFC 1750. So far as I know, RFC 1750 has been the best source of advice to developers who need

entropy sources and do not have hardware devices that have been designed to provide it explicitly.

As with all RFCs, this document is freely available for reproduction and distribution If NIST feels the need for a formal release, we'll be happy to sign one.

I hope you find this helpful, and I wish you great success with the workshop. This is a pivotally important topic.

Cheers,

Steve

From: "J. Andrew Rogers" <andrew@ceruleansystems.com>
Subject: Comments on X9.82
Date: Sun, 27 Jun 2004 09:46:46 -0700

As a mathematician who has spent many years working in algorithmic information theory and computational models that deal with the predictive limits of finite state machinery, I was somewhat puzzled by some of the assumptions used in the specs I saw.

The practical distinction between DRBG, NRBG, and pseudo-random appears to be at odds with some areas of theoretical mathematics that are relevant to the discussion. From the standpoint of finite state machinery (FSM), there is no theoretical distinction between "pseudo-random" and NRBG because both are defined in terms of the predictive limits of the FSM. From a strict computational perspective, there is no justification to ever assume one or the other because for any reasonably strong DRBG these two will be indistinguishable on all hardware in our universe. And this should be quite provable in many cases. In other words, every RBG should be presumed to be pseudo-random because even apparently statistically perfect physical entropy sources will probably never be provable as anything else. I'm probably not the first person to make this general point, but the crux of my point revolves around randomness being generally defined in terms of the predictive limit in current algorithmic information theory, which is a well-grounded definition generally applicable here and which has some interesting consequences.

Which brings me to my other point.

While the general mathematical test of randomness, universal sequence prediction, is not generally scalable above bit patterns greater than a few dozen bits, it would seem reasonable to apply thorough analysis of RBG algorithms using this test. The problem with this test is that even though it is exhaustive, it is limited to low-order information theoretic patterns on real hardware. But far less known, and known to me primarily because it is one of my core areas of research, is that there are a number of extremely good approximations of Solomonoff induction (i.e. "universal predictors") that are capable of discerning anisotropies in high order information from a streams of nominally "random" bits with no readily apparent low-order anisotropies (and therefore apparently "random"). While I only skimmed through the standard, the verification process for both NRBG and DRBG seems to be thin on good mathematical validation that implementations are what they say they are, at least to the extent that we can test such things with mathematical metrics.

I don't have anything invested in this, since crypto and RBGs are not something I ever normally deal with, but someone not associated with the current process forwarded the X9.82 working papers to me for an opinion which I felt worth forwarding to the NIST. As such, I may be chasing a tangent here for the purposes of this NIST standard, but there is a chance that you may find my comments helpful. :-)

Cheers,
j. andrew rogers

From: "Walker, Jesse" <jesse.walker@intel.com>
To: "Elaine Barker" <elaine.barker@nist.gov>
OriginalArrivalTime: 28 Jun 2004 16:48:46.0530 (UTC)

We have a question concerning the documents posted for the workshop. X9-82_Part3_workshop.pdf is one of these, and its Clause 10.2.1 on page 108 says that

> A DRBG based on block ciphers uses a key in conjunction with each block cipher algorithm used by the DRBG and an initial value. Section 10.3.2 specifies a DRBG that ....

The ellipsis & is included in the text, so the sentence is incomplete. It seems to be a fragment pointing to Clause 10.3.2 as the location of an approved block-cipher-based DRBG. However, looking at page 109 shows that Clause 10.3.2 defines an ECC-based DRBG instead of a block-cipher-based DRBG. In reviewing the remainder of the document we could not find a block-cipher-based DRBG. We expected to see one based on AES. Is this omission intentional? Any insight would be appreciated.

-- Jesse Walker

From: “Martin Kretschmar” <mail@martin-kretschmar.de>
Subject: [IP] Standard Random
Date: Mon. 28 June 2004 16:52:26 +0200

I would like to point out at http://www.lavarnd.org/ and
http://www.math.keio.ac.jp/~matumoto/emt.html.

Regards,
Martin Kretschmar

From: "Weis, Steve" <sweis@rsasecurity.com>

First off, I think the current version of part 2 of the draft is primarily oriented towards PCs and or other large platforms. Although it may be outside the requirements of this standard's intended audience, I think it is worth considering some of the issues associated with implementing RNGs in low-cost devices like RFID, sensor networks, or basic smartcards.

In these settings, power and gate counts may be highly limited. Devices may not have an internal power source, but rather will rely on harvesting energy via RF signals. It may not always be possible to update local persistent state.

In the context of the X9.82 specification, these devices may only have an entropy source and entropy conditioning logic - essentially the most basic NRNG defined in the specification. It is unlikely that they will have self-testing capabilities or a fallback deterministic RNG. Any RNG health diagnostics will need to be conducted by an external device. Something like the simple ring oscillator design proposed yesterday by Doug Whiting might be appropriate for an RFID device or a sensor network mote.

There is a growing need for low-cost RNG designs, particularly for RFID devices. These designs would benefit greatly from a basic NRNG or entropy source specification, flexible enough for low-power and low-gate count environments.

I hope these comments are useful. Thanks again for hosting the workshop.

Modern Methods and Applications of Random Number Generation in Signal Processing

Devin Cambridge
Cambridge and Smith, LLC
4735 Sepulveda Blvd. #123
Sherman Oaks, CA 91403
(650) 933-3386

Devin.Cambridge@cambridgeandsmith.com

ABSTRACT

In this paper, we examine the sources of random numbers used in signal processing. We also hope to present some interesting solutions to modern application problems using random numbers and **to** provide methods in which to test the integrity of random number sequences for use in a variety of applications.

General Terms

Random Number Generator (RNG), Pseudo-Random Number Generator (PRNG). FIPS 140-1.

Keywords

Random Number Sequence Flavors, Normal RNG, Attractive RNG, Ideal RNG, Hybrid RNG, Multi-body Initialization, Simulation Pavlovian Response.

1. INTRODUCTION

Randomness exists as a problem in the electrical engineering world. Most of the time, electrical engineers try to minimize fluctuations, discrepancies and the effects of the chaotic world. Designing randomness into a systematic process on a machine adhering to logic operations and designed to maintain a high level of order seems counterintuitive. However in the modern age of high transaction and high frequency computing, the desire to create a more organic interaction with electronics as well as a heightened awareness of the need for security has driven application designers to take a second look at the nature of randomness.

2. Random Number Flavors

In the past, engineers and mathematicians looked at random number sequences as if they were all the same. In fact, there are many different flavors of random number sequences. The three we will focus on are:

Normal or initialized random number sequences are those which begin at zero but do not converge to any finite value at infinity, do not exhibit linearity in N dimensional space, do not repeat at any appreciable time increment, and any number along the sequence cannot be calculated at time $T= (T_0 + \Delta T)$

Attractive or synchronized random number sequences are those that begin at zero, do not converge to any finite value at infinity, may exhibit some linearity in N dimensional space, do not repeat at any appreciable time increment, and any number along the sequence might be calculated at time $T= (T_0 + \Delta T)$

Ideal or true random number sequences exhibit all the characteristics that most mathematicians would qualify as random. They do not converge to any finite value at infinity or negative infinity, they do not exhibit linearity in any N dimensional space, they do not repeat at any appreciable time increment, and any number along the sequence cannot be calculated at time $T= (T_0 \pm \Delta T)$

3. Sources of Random Numbers

There are two classical sources for random numbers, physical events and mathematical pseudo-random number generators. Both possess advantages and disadvantages.

3.1 Physical Sources

Physical phenomena seem to provide a good place to start when looking for a source. Many engineers choose to sample physical events and have devised elaborate methods to do so. As mentioned in [9], engineers have used everything from interrupt events on computers to radioactive decay measurements as sources of random numbers. The most common source comes from thermal noise [4; 8].

However, sampling of physical events can be problematic. For one, random bits must be distilled from the physical events. The resultant operations to ensure a truly random sequence lengthen the time of acquisition and add to the overhead of the operation. Thus, while the raw physical source may provide adequate randomness for an application, the time period may seriously impede application performance. Secondly, every source must be sampled and errors in sampling, such as aliasing and quantization noise can transform the original waveform adding unwanted results. Thirdly, as time goes on, the understanding of physics grows and the modeling of physical systems becomes more precise. This gives physicists more of an advantage towards the inherent biases in physical systems and the ability to gauge the pertinent initial conditions used in a sample system. Finally, the greatest weakness inherent in a physical system remains the ability of all physical systems to be driven. Figure 1 shows a Gaussian noise source simulating a sampled thermal environment.

ISPC conference '03, March 31-April 3, 2003, Dallas, Texas.

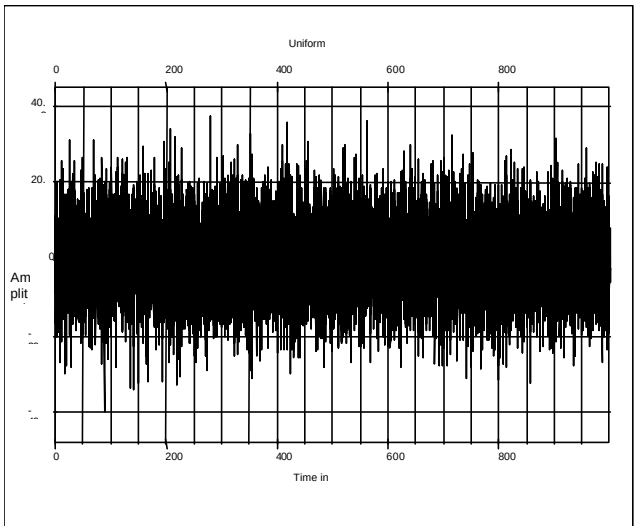


Figure 1. Sampled Gaussian (thermal) noise [Std. Dev 10e-3 V; Sampled at 10Khz] – *Graphs generated with Elanix SystemView*

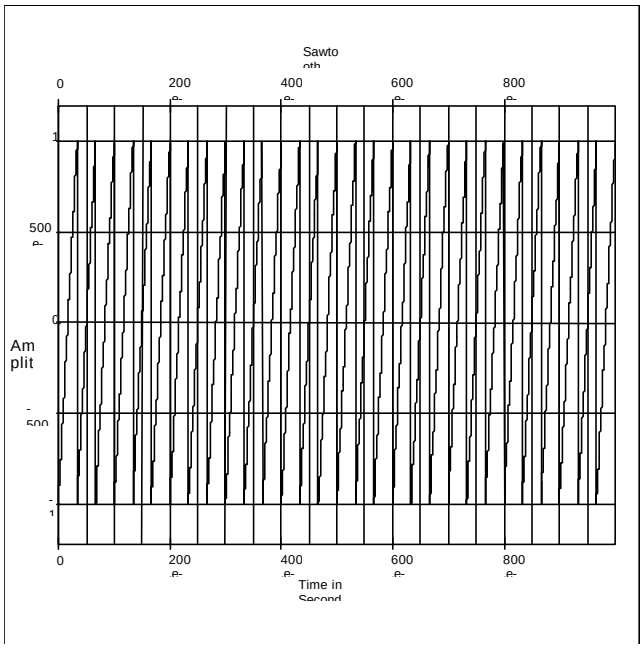


Figure 2. Driving System (Sawtooth wave) [Amp 2v; 30HZ]

We can simulate what would be an attack on the randomness of the system by superimposing a periodic wave onto our thermal sample. Figure 3 show that if the driving system has enough power and greater frequency than the thermal noise, we can drive the system into a set periodicity. We should note that many times these attacks happen not at the source of the physical data but at the sampling device itself by inductance or other means.

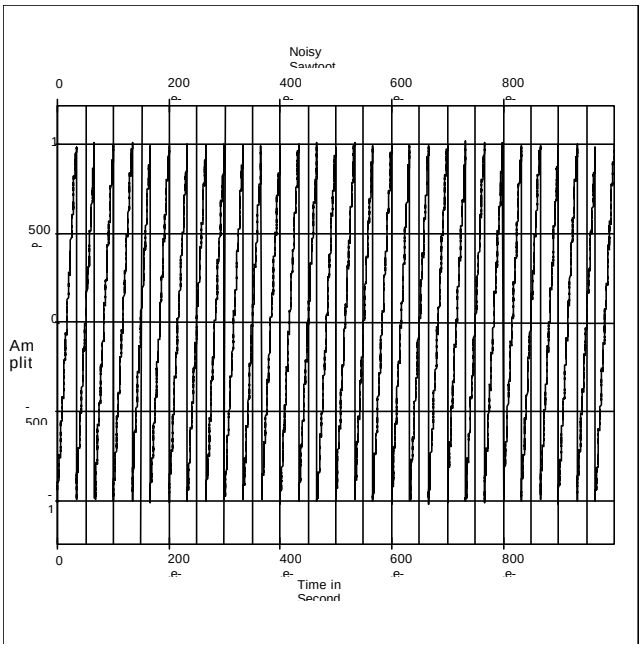


Figure 3. Sampled Gaussian Noise + Driving System

A recent example of this form of assault was the attack on an old version of the UNIX /dev/random device. This Unix device file provided random numbers by sampling system interrupts from various input devices on the machine. However, when a machine acts as a server with infrequent mouse, keyboard or input device interrupts, the primary interrupt comes from the disk access. By forcing the machine to swap heavily, the random pool became corrupted.

3.2 Mathematical Pseudo-Random Number Generators

Mathematicians and statisticians supply the electrical engineering world with its most common tool for generating random numbers, mathematical equations designed to provide a string of numbers that exhibit normal or attractive properties. These numbers are ideal in that they can provide random bit streams quickly and sometimes near the frequency of the processor.

However, these systems also bear inherent problems. One such problem comes from bad equations. Only recently have researchers started to look at the quality of the equations used in random number generation on high register bit, high frequency processing. Also, designing systems with long-term random integrity remains difficult and requires a fairly in-depth knowledge of mathematics. Another problem comes from the need to initialize or seed the system. Engineers used static seeding in the past by choosing a random number to initialize the system. As the system ran, the sequence played out. If the system reinitialized, a replay of the sequence would start and compromise the non-predictability of the system. A shortcut commonly used to seed a system by DSP and electrical engineers remains tapping the swap file, registry or memory values. However, these values are not truly random and are easily corrupted by outside influences looking to again drive the system. In addition, Knuth [5] discusses the need to reject certain seeds as biased.

3.3 Hybrid RNG Systems

One of the best methods of overcoming the limitations of a pure physical system and a mathematical pseudo random system is to combine the two and use their combined strengths to overcome the inherent weaknesses of each individual system.

3.3.1 Direct Physical Seeding of an PRNG

One classical hybrid method uses data from a physical source RNG to seed a PRNG. This method serves best when using one PRNG and seeding it with the physical RNG at initialization *as multiprocessing systems tend to have problems with this method.* While this system overcomes the initialization vulnerability inherent in a PRNG, note that should the physical sampler be deactivated, the application could hang or be statically seeded. To overcome this, the use of an entropy pool made from physical samples (or a list of manually entered seed values) can be used. However, it should be reiterated that ideally random number strings should be created in real-time and not stored in a place that could be compromised. Moreover, this system still possesses the vulnerability to physical driving, but only the seed value will be affected providing a limited advantage over a pure physical system.

3.3.2 Using a Cryptographic Hash

Cryptographic hash functions produce output of a fixed length from any size input. Also, hash functions possess the unique ability to change the output signature dramatically based off of a small change in input. Many current popular cryptographic algorithms such as DES and AES incorporate hash functions as secondary modes of operation. The hash function provides us with a quick way of generating random numbers from a physical source without the overhead necessary in distilling the physical source directly.

A better design would be to use a hash of a physical source to seed a PRNG. U.S. Patent number 5,732,138 ``Seeding a pseudo-random number generator with the cryptographic hash of a digitization of a chaotic system" gives an excellent example of this. This patent gives an example where a CCD digitizes a system of lava lamps. The resultant image is then hashed and used to seed the random number generator. DSG [1] provides a nice analysis to the effectiveness of this design. While the patented system is bulky, sleeker designs using the patent are possible. Figure4 shows a general design for a hybrid device based on the previously mentioned patent, using a small CCD susceptible to noise. An initializing picture is taken, verified by the DSP, and stored in flash RAM. All subsequent pictures are then superimposed with the original in flash RAM. The superposition of the initializing image, the secondary images, and thermal noise should provide enough difference to create widely varying hash values by the DSP. If the secondary image should become dark or be burned out, the image in the flash ram combined with any latent noise should be able to provide the DSP with enough interesting data to hash. When using physical samples as hash input, the bit range should be at least 2x the hash bit size. Thus, if the hash output size is 128 bits then at least 256bits of data should be the input size to ensure that the range of possible outputs will be met and that the likelihood of a collision will be reduced.

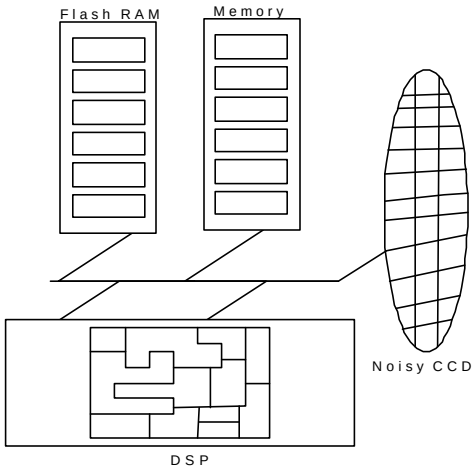


Figure 4. Sample Hybrid RNG

4. Applications

RNG applications have advanced since the early days when simulations merely replicated a card shuffle or a dice role. Jumping to the 21st century, random sequencing demands have increased as users interact with each other on massive scales. Moreover, consumers today engage in more media rich content on their computers than they did at the advent of home computing. High end graphics card with 3D particle rendering, DVD players with interframe compression, heuristic desktop interfaces that customize themselves based off of past user selections and interactive dictionaries that correct spelling in real time are all now part of the common user experience. And RNGs are playing a larger role in giving the end user a more organic computer experience. On the high end are large simulations which expect computers to do a large volume of computations in time frames unfathomable years ago.

Thus, with the advent of high speed, mobile computing, the usefulness of RNGs entered the light. Intel in a 1999 Paper [3] listed five categories for applications using RNGs: *Entertainment, Music and Graphics Composition, Simulation (including AI) and Testing, Equation-Solving, and `Cryptography, Digital Signatures, Protected Communications'.* However, the application of a RNG in these applications is not always as straightforward as it seems.

4.1 Multi-body initialization

In advanced 3D graphics, RNGs provide the perturbations to particle effects as well as initialization to large-scale particle systems. Due to the work by Henry Poincaré in 1903 and later by Edward Lorenz in 1984, we know that large body systems will progress differently depending upon the initial conditions. By utilizing a RNG, the initial parameters such as particle placement can be can be quickly generated to give a more organic feel to fog and particle effects. The dependency on initial conditions allows the user a real world feel over the linearity produced by a computer. In this case RNG's have a function to speed up the initialization time.

4.1.1 Problems with Multi-body RNG initialization

One issue with using an RNG to initialize the state of a multi-body problem comes from collisions in the RNG output. For this reason, it is suggested that a hash function be used. Care must be taking in choosing an RNG and hash function that will avoid placing two objects at the same position. If the resources are available, a small database can be used to eliminate collisions. However, this will slow down the initialization process. A better method requires starting with a linear matrix of particles and using the RNG as a perturbation to the placement of the particles.

4.2 Simulation Artificial Intelligence

Simulations play a large role in both the commercial and the industrial space. Perhaps the most interesting use of simulations involves Artificial Intelligence (AI). RNGs greatly help in AI decision making, allowing it to overcome simple loop functions and providing different pathways for objects to move.

4.2.1 Simulation Pavlovian Response (SPR)

One of the issues with using Normal and Attractive RNGs comes from the re-initialization effect. In game play, humans possess strong pattern recognition ability. Any sequence that repeats itself often registers subconsciously by the human participant. If the RNG possesses a static seed, when the sequence reinitialized, it will repeat the same pattern. If a human participant registers the pattern, they interact with the simulation at a reflexive level. At this stage, the human is trying to beat the simulation as opposed to beating the AI opponent. In advanced simulation, this can be devastating. For example, take a fighter simulation. In our example, the eager pilot has scheduled to be the first person to use the simulator each morning for week. On day one, in a one on one engagement, the pilot performs a series of maneuvers and gets hit on the twentieth maneuver. The next day, our pilot tests on the same scenario. Again, they are hit on the twentieth maneuver. The next day, our pilot is shot down on the fifty-second maneuver. What happened? The pilot learned the AI pattern up to the twentieth maneuver due to the fact that the simulator was shut down each night and reinitialized with the same seed the next morning. At this point, it should be questioned if the pilot is learning to react to the opponent AI or if they are just memorizing a pattern.

It should be noted that SPR could be beneficial in a simulation. For example, in commercial gaming applications for pure entertainment, sometimes using two different RNGs, one that has a pattern and one that doesn't can provide a medium and high difficulty setting. Also, sometimes simulations are specifically designed to impart SPR. Such examples of this would include driving simulators designed to control a skid or a flight simulator designed to teach how to deal with a microburst. However, most of these situations are better handled with a control loop and not RNG.

4.3 Cryptography

With the ascendancy of the Internet, cryptography leads all applications in the use of RNGs with its applications in authentication and secure communication. Because of its nature, cryptographic use of RNGs attracts the most scrutiny. And, it should. The future of licensing, privacy and e-commerce rest on strong cryptography.

4.3.1 Problems in E-commerce – Credit Card Numbers

Credit cards present a big problem to the financial model of e-commerce. One of the main issues with credit cards comes from the sequence used to generate the next replacement card. Past problems have been compromises of the card sequence by reverse engineering the RNG algorithm used to create the new card.

4.3.2 Channel Security

Network Communications use PRNGs in ways that are not always obvious. TCP/IP uses PRNGs to generate the Initial Sequence Number (ISN) for channel communications between clients and hosts. According to Michal Zalewshki [10], a theoretical spoof attack based on guessing the ISN sequence originally was proposed in 1985. Zalewshki did further work in 2001 to analyze various OS PRNGs to gage their susceptibility. The results are surprising as some of the major operating systems exhibit strong attractive properties in their RNG implementations.

5. Statistical Tests

So now that some of the problems with bad number sequences have been identified how do we determine what constitutes a good RNG?

5.1 FIPS 140-1 Statistical Random Number Generator Tests

The U.S. Department of Commerce set a series of standards for cryptographic use. Part of this standard involves statistical tests for RNGs. According to FIPS 140-1 RNGs are considered adequate if they pass the following criteria:

" A single bit stream of 20,000 consecutive bits of output from the generator is subjected to each of the following tests. If any of the tests fail, then the module shall enter an error state.

- *The Monobit Test*

    1. Count the number of ones in the 20,000 bit stream. Denote this quantity by X.

    2. The test is passed if $9{,}654 < X < 10{,}346$.

- *The Poker Test*

    1. Divide the 20,000 bit stream into 5,000 contiguous 4 bit segments. Count and store the number of occurrences of each of the 16 possible 4 bit values. Denote *f(i)* as the number of each 4 bit value *i* where $0 <= i <= 15$.

    2. Evaluate the following:

$$X = (16/5000) * \left(\sum_{i=0}^{15} [f(i)]^2 \right) - 5000$$

3. The test is passed if $1.03 < X < 57.4$.

- *The Runs Test*

1. A run is defined as a maximal sequence of consecutive bits of either all ones or all zeros, which is part of the 20,000 bit sample stream. The incidences of runs (for both consecutive zeros and consecutive ones) of all lengths (>= 1) in the sample stream should be counted and stored.

2. The test is passed if the number of runs that occur (of lengths 1 through 6) is each within the corresponding interval specified below. This must hold for both the zeros and ones; that is, all 12 counts must lie in the specified interval. For the purpose of this test, runs of greater than 6 are considered to be of length 6.

Length of Run	***Required Interval***
1	2,267-2733
2	1,079-1,421
3	502-748
4	223-402
5	90-223
6 +	90-223

- *The Long Run Test*

1. A long run is defined to be a run of length 34 or more (of either zeros or ones).

2. On the sample of 20,000 bits, the test is passed if there are NO long runs."

5.2 The Trinity College Tests

In 2001 researchers at Trinity College [1] in Ireland chose from a serious of theoretical and empirical tests for RNG's to run against some of the Internet RNGs. While [1] goes in depth into the runs, the suite they chose were:

- *A chi-square test*
- *A test of runs above and below the median*
- *A reverse arrangements test*
- *An overlapping sums test*
- *A binary rank test for 32x32 matrices*

5.3 Diehard Code

Florida State University, under a grant from the U.S. National Science Foundation, Gail Gasram wrote a serious of computer functions for various operating systems to test random numbers. At the present time, they can be found here:

ftp://stat.fsu.edu/pub/diehard/

6. REFERENCES

[1] Distributed Systems Group, CSD, Trinity College Dublin, ANALYSIS OF AN ONLINE RANDOM NUMBER GENERATOR. April 2001 http://www.random.org/report/Report.pdf

[2] FIPS 140-1, SECURITY REQUIREMENTS FOR CRYPTOGRAPHICS MODULES, Federal Information Processing Standards Publication 140-1, U.S. Department of Commerce/NIST, National Technical Information Service, Springfield Virginia, 1994

[3] INTEL CORPORATION, THE INTEL RANDOM NUMBER GENERATOR. ftp://download.intel.com/design/security/rng/techbrief.pdf.

[4] Johnson, John Bertrand. Electronic Noise: The first two decades. IEEE Spectrum 1971, pp 42-46.

[5] D. Knuth, The Art of Computer Programming: Volume 2, Seminumerical Algorithms, 2nd edition, Addision-Wesley, 1981

[6] Maurer, U. M . " A Universal Statistical Test for Random Bit Generators," Advances in Cryptology- CRYPTO '90 Proceedings, Springer-Verlag, 1991, pp 409-420

[7] Maurer, U.M. " A Universal Statistical Test for Random Bit Generators," Journal of Cryptology, v. 5 n.2, 1992, pp. 89-106.

[8] Sklar, Bernard. Digital Communications: Fundamentals and Applications. Prentice Hall Inc., Upper Saddle River, New Jersey, 2001

[9] Schneier, Bruce. Applied Cryptography: Protocols, Algorithms, and Source in C, John Wiley & Sons, Inc, 1996

[10] Zalewski, Michal. Strange Attractors and TCP/IP Sequence Number Analysis, BindView Corporation, 2001. http://razor.bindview.com/publish/papers/tcpseq/print.html

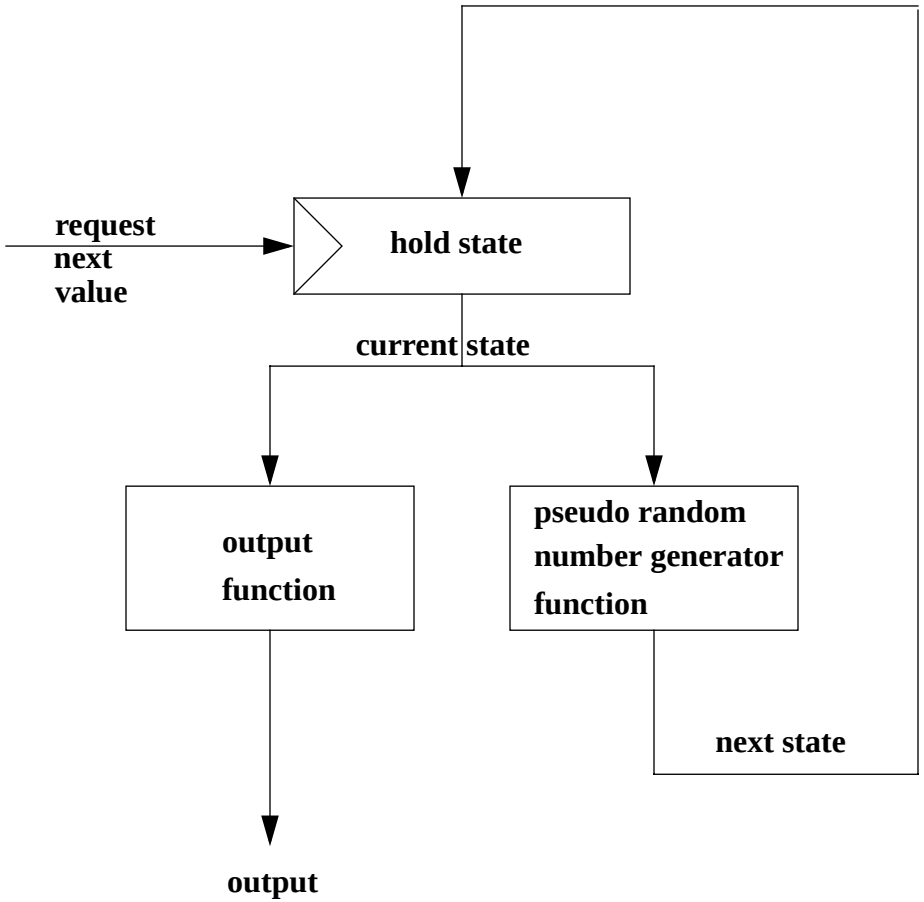
Columns on Last Page Should Be Made As Close As Possible to Equal Length

CONTINUOUS RANDOM NUMBER GENERATION

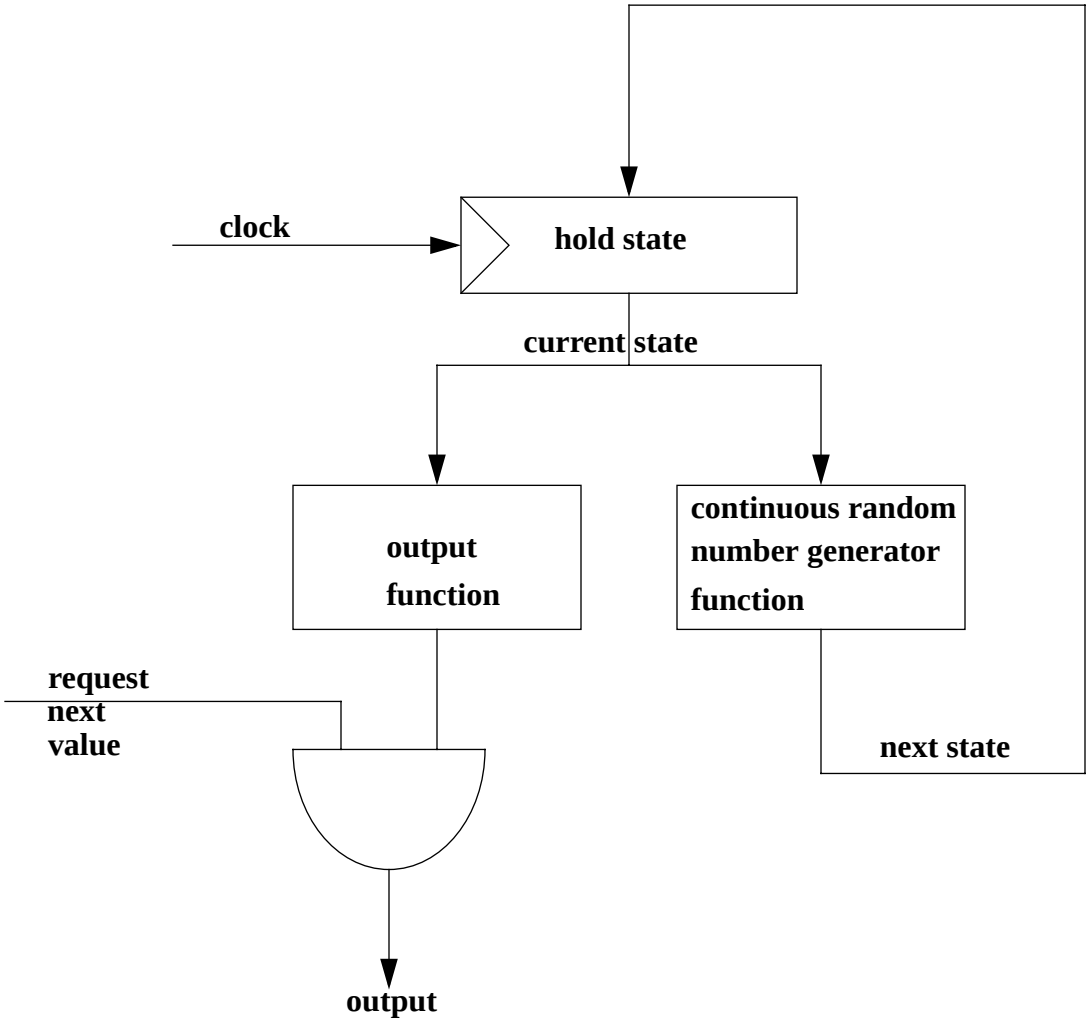
LEONARD RARICK

SUN MICROSYSTEMS

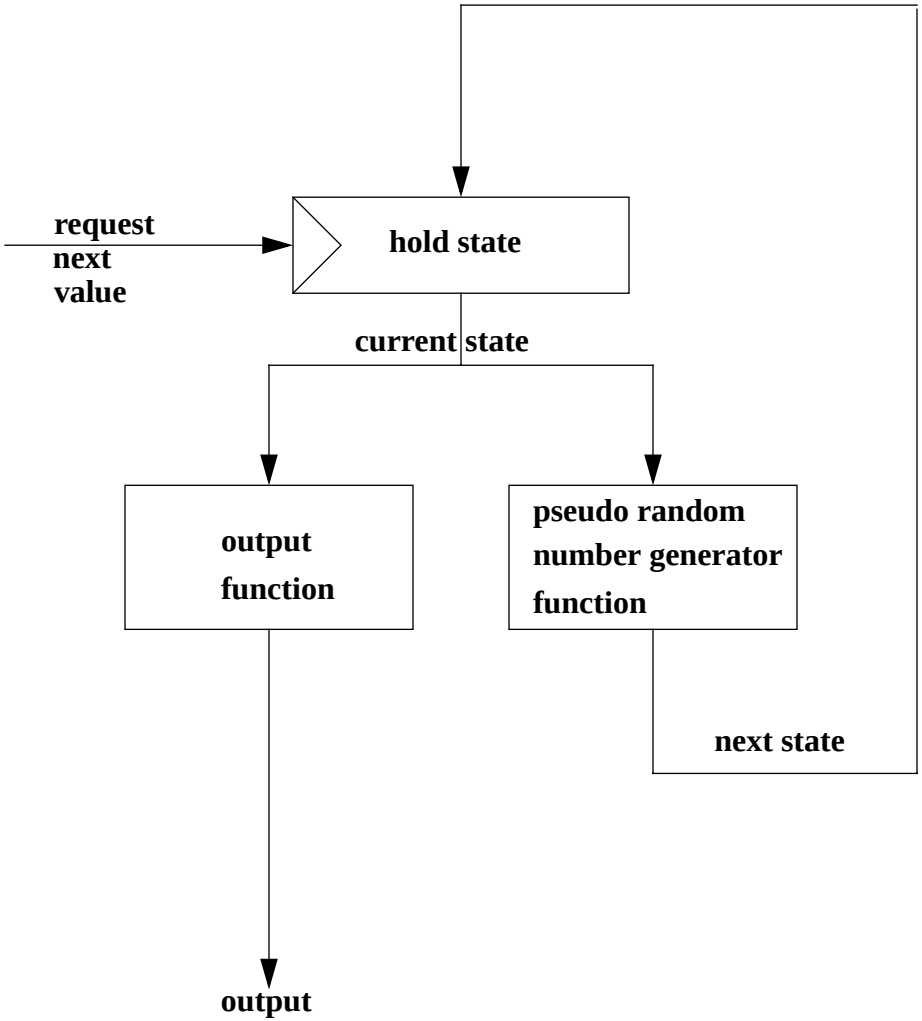
PESUDO RANDOM NUMBER GENERATOR



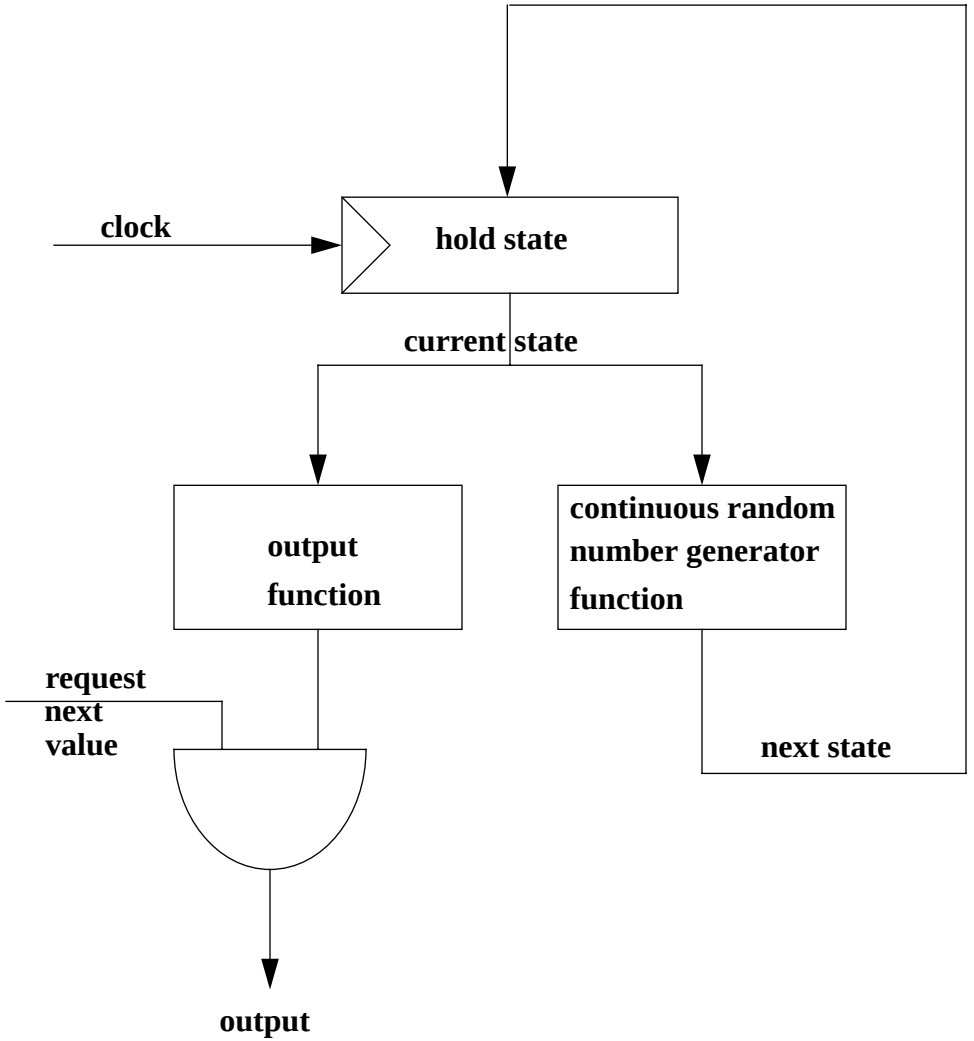
MINIMAL CONTINUOUS RANDOM NUMBER GENERATOR



PRNG

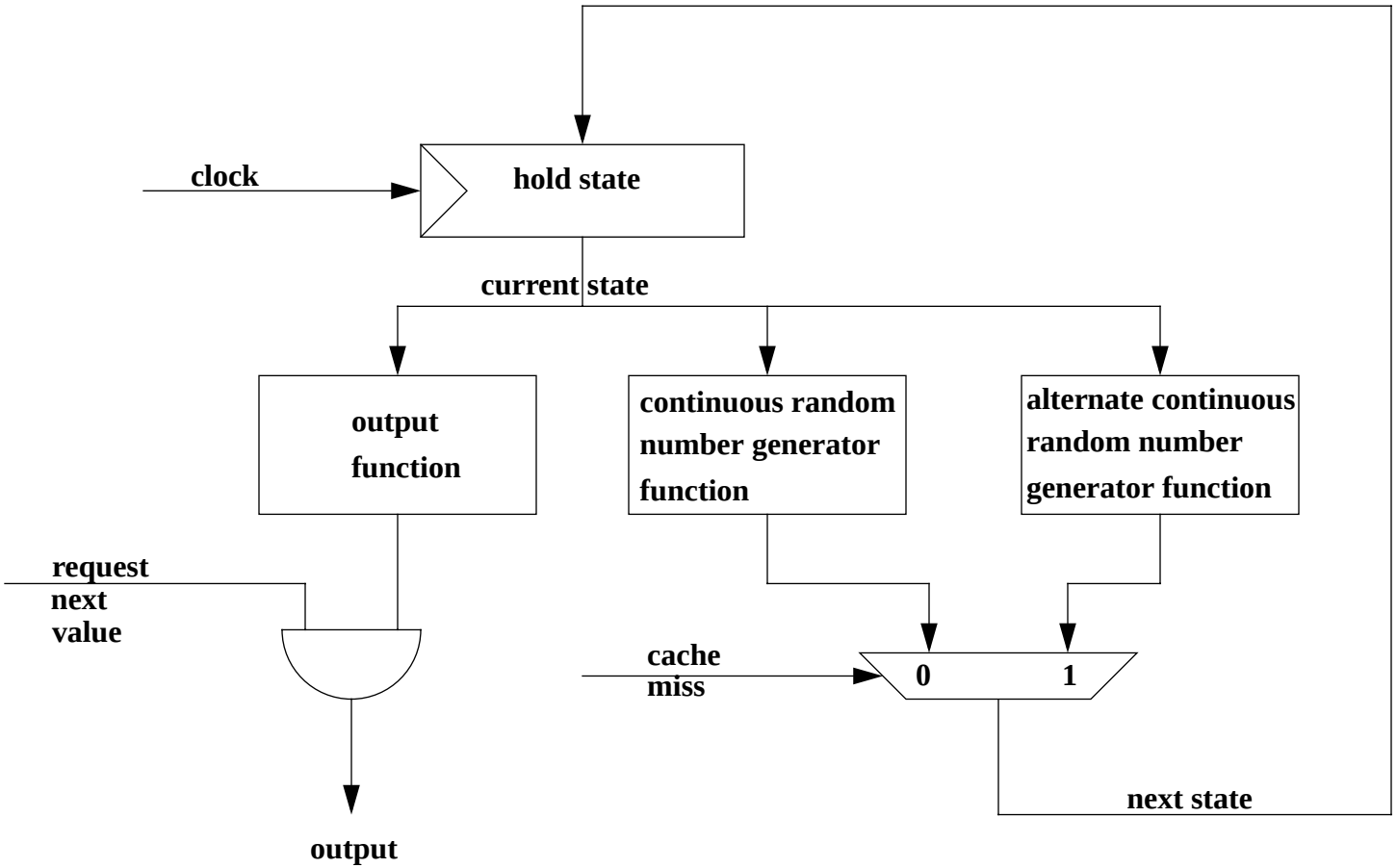


CRNG

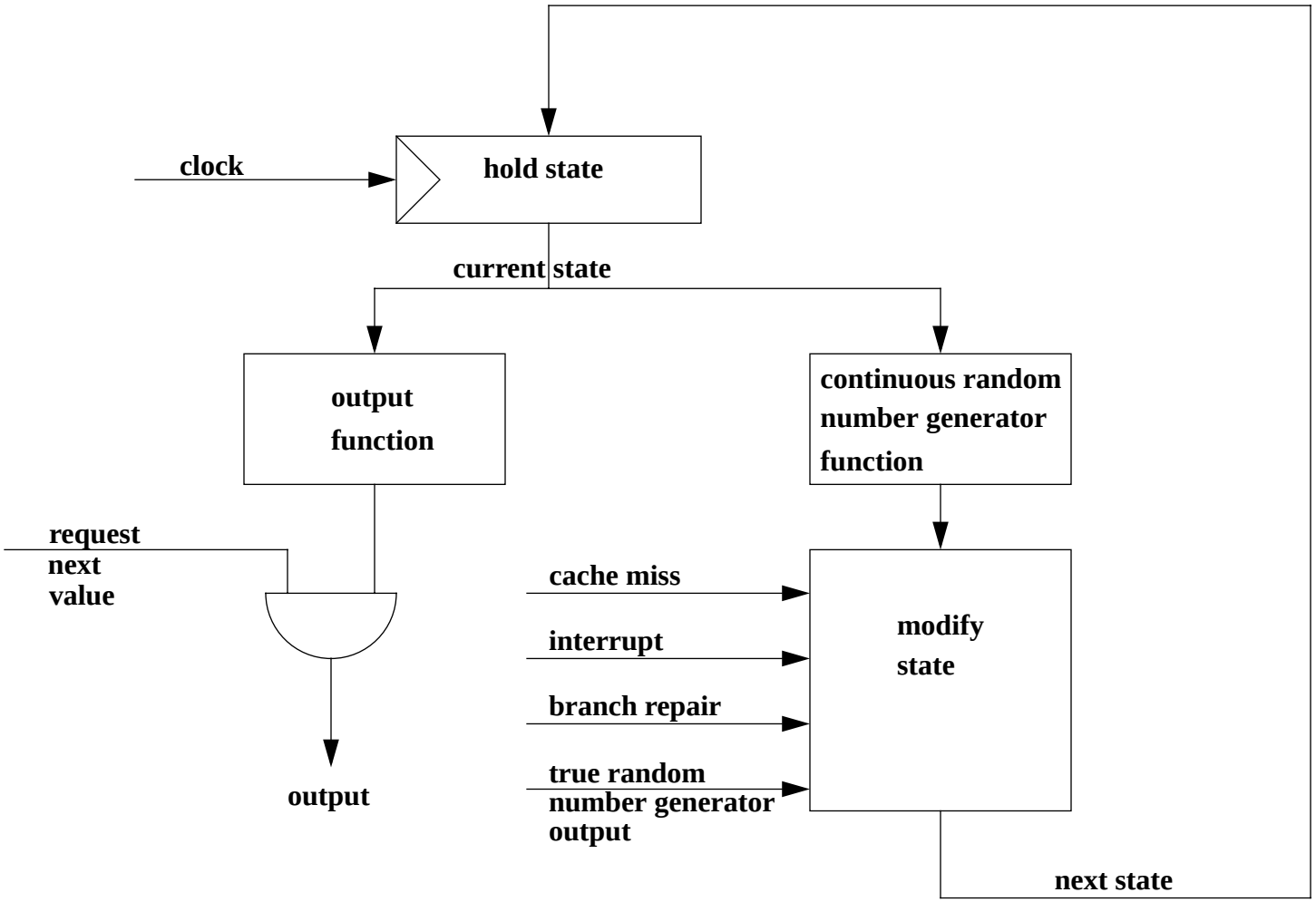


Sun microsystems

USING A SOURCE OF ENTROPY

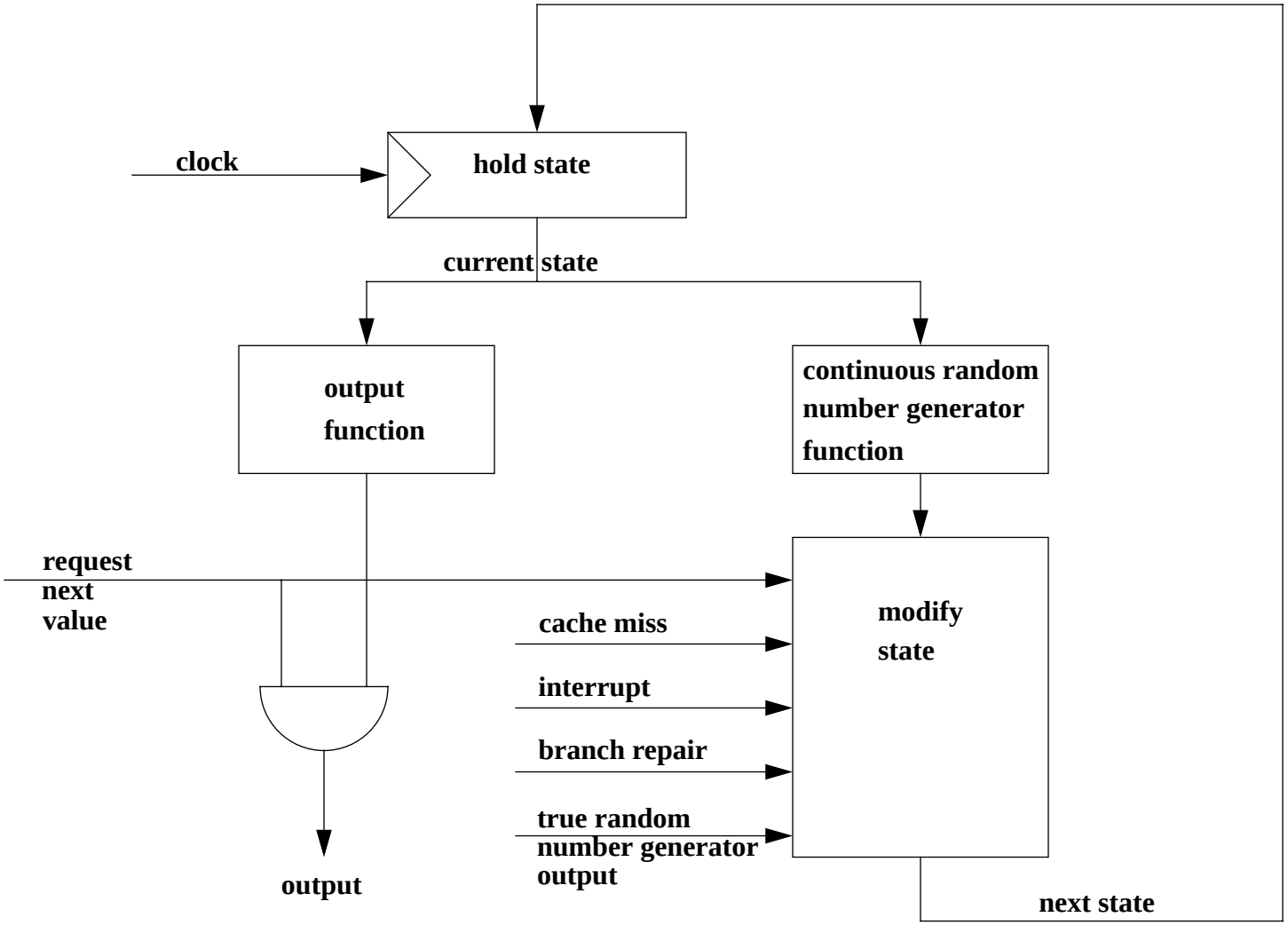


USING SEVERAL SOURCES OF ENTROPY



Sun microsystems

HEISENBERG RANDOM NUMBER GENERATOR



Sun microsystems