CPE 462
VHDL: Simulation and Synthesis

Topic #09 - a) Introduction to random numbers in hardware

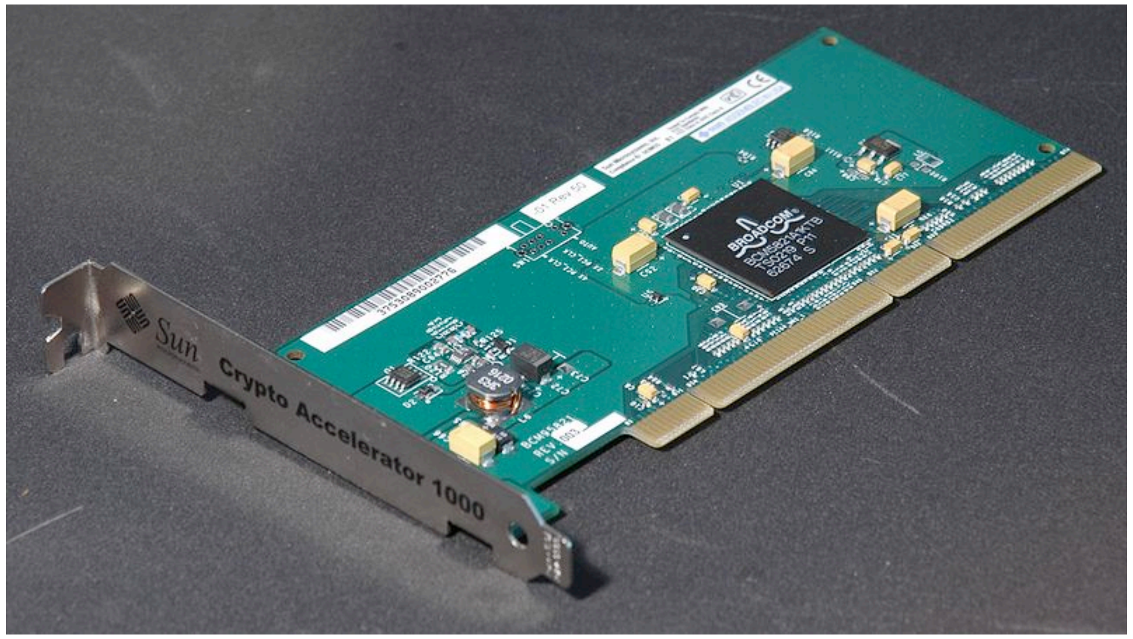
Nuno Alves (nalves@wne.edu), College of Engineering

Dürer's engraving of Fortuna (1502)

Fortuna was the goddess of fortune and personification of luck in Roman religion. She might bring good luck or bad: she could be represented as veiled and blind.

Three ways to generate random numbers

- Harvesting random signals in hardware
- Using a software algorithm based on a seed
- Using "random" macroscopic processes



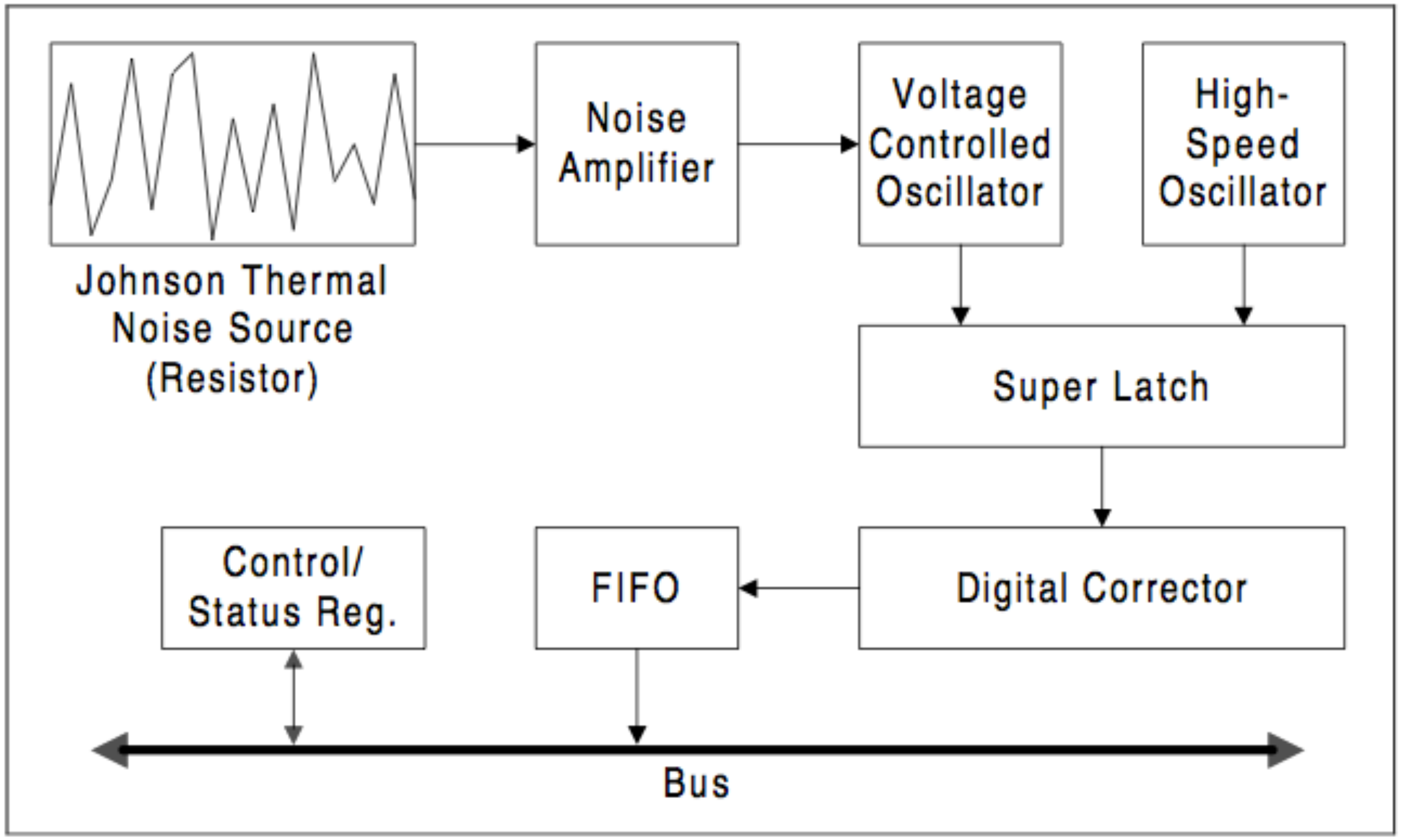
- Why do we care about randomness? Secure communication, simulation of physical events, games, ...

Random numbers generated by hardware

Hardware random number generator

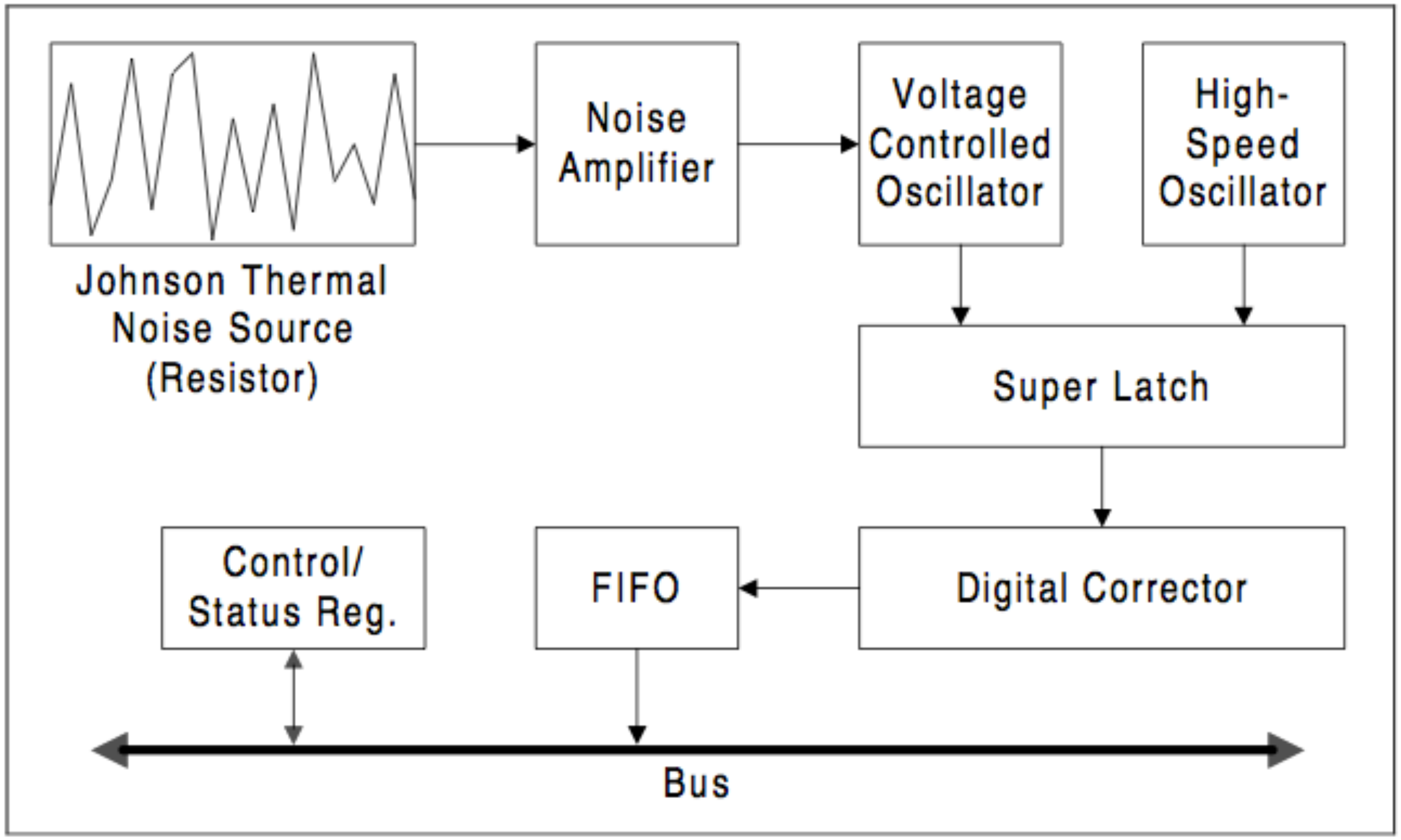
- What is a Hardware random number generator ?
  - Is an apparatus that generates random numbers from a physical process.
- How can you create random numbers in hardware?
  - Often based on microscopic phenomena that generate a low-level, statistically random "noise" signal, such as thermal noise, nuclear decay, the photoelectric effect or other quantum phenomena.

Intel Random Number Generator



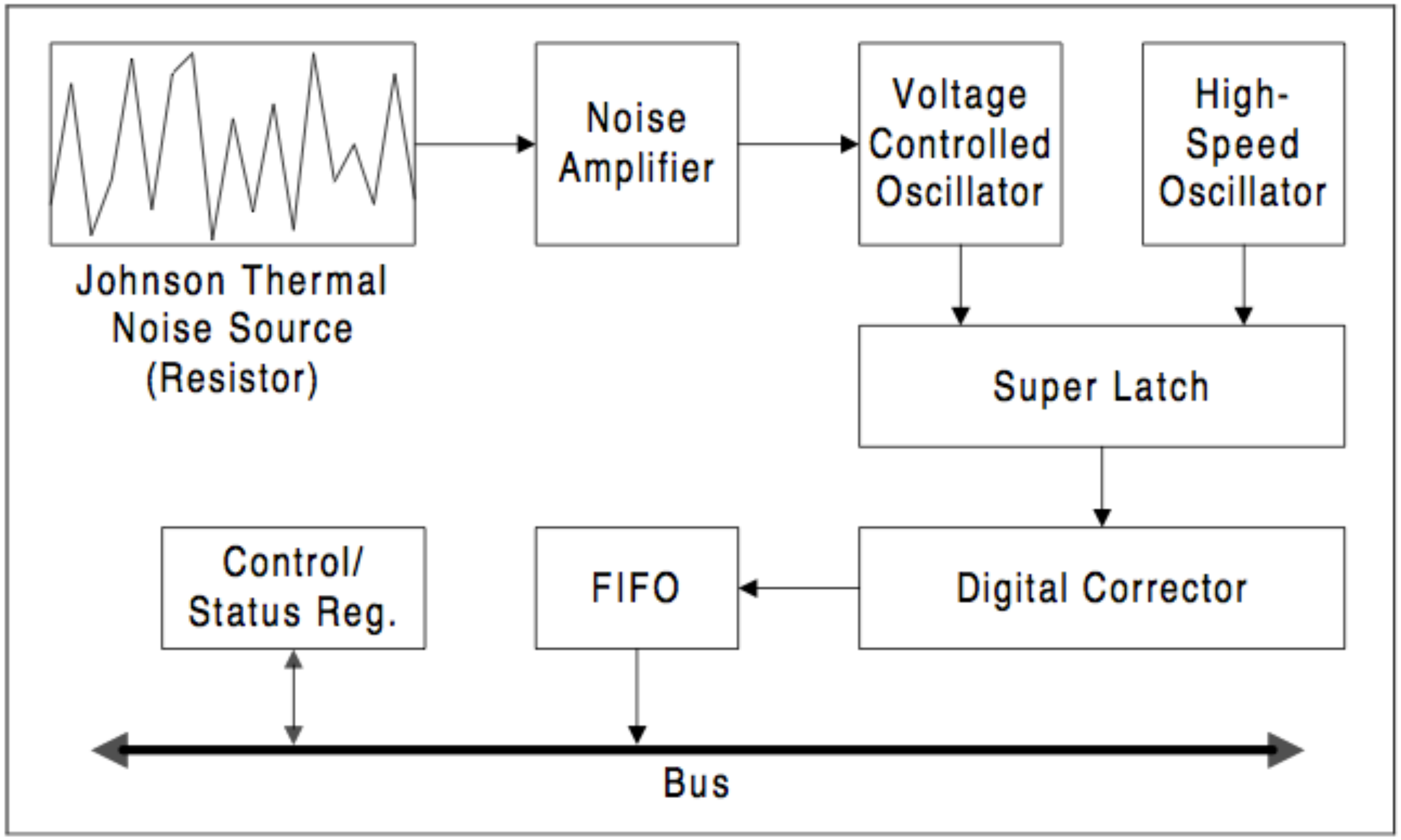
- Johnson noise (or thermal noise), shot noise, and flicker noise are present in all resistors.
- They have electrically measurable characteristics and are the result of random electron and material behavior.

Intel Random Number Generator



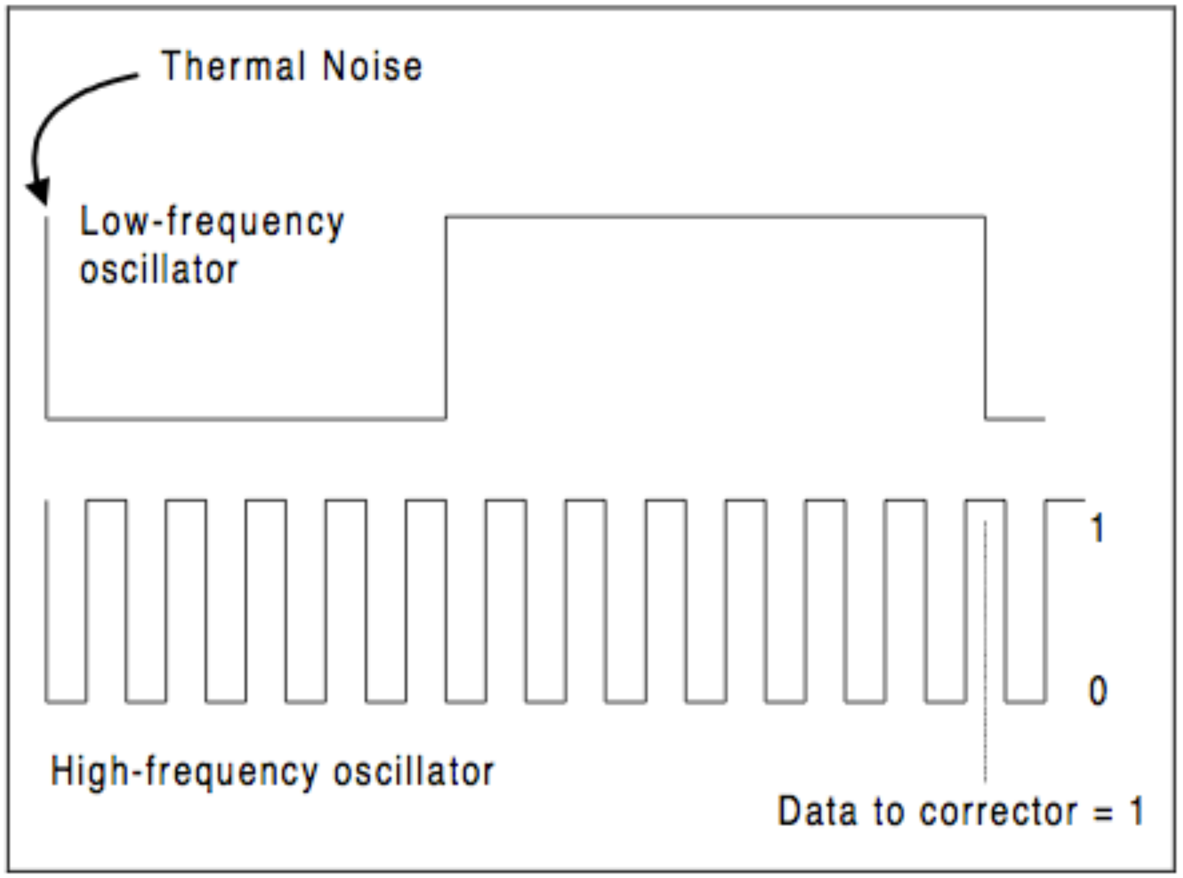
- The Intel RNG primarily samples thermal noise by amplifying the voltage measured across undriven resistors.
- These measurements are correlated to local pseudorandom environmental characteristics, such as electromagentic radiation, temperature, and power supply fluctuations.

Intel Random Number Generator



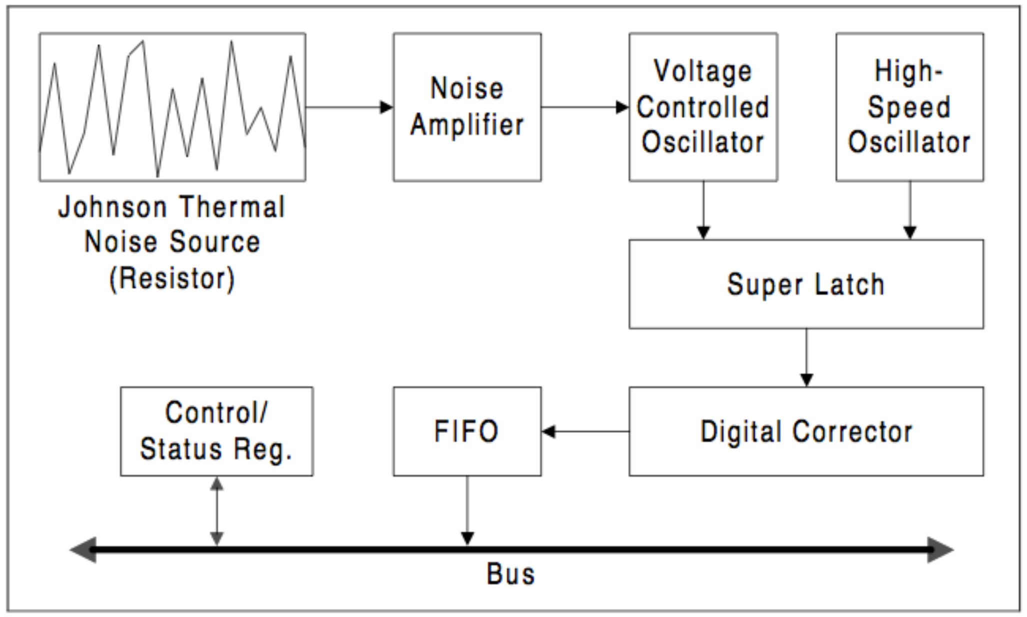
- The Intel RNG significantly reduces the coupled component by subtracting the signals sampled from two adjacent resistors.
- Circuitry used for amplification or signal handling should preserve as many of the components of randomness as possible

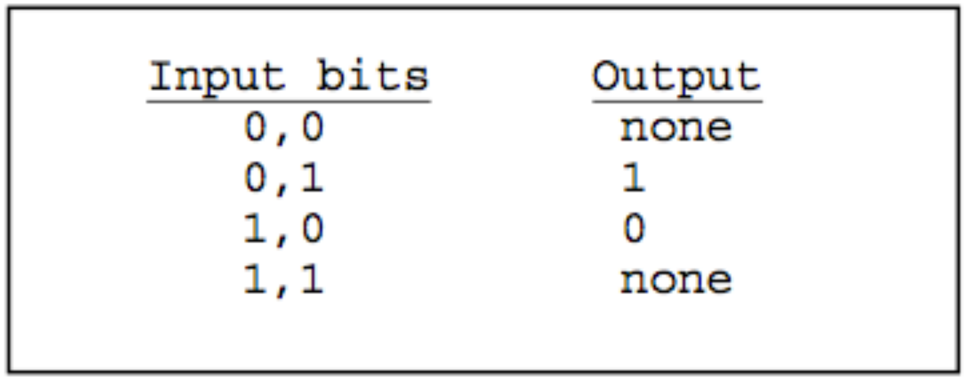
How does it actually work?



- You have two oscillators: one very fast and one very slow.
- Thermal noise adjusts the frequency of the slower clock.
- The varying, noise-controlled slower clock is used to trigger measurements of the fast clock.
- Drift between the two clocks thus provides the source of random binary digits.

Von Neumann Corrector





Input bits	Output
0,0	none
0,1	1
1,0	0
1,1	none

It cannot assure randomness in its output, however it will transform a biased random bit stream into an unbiased one.

- The initial random measurements are processed by a hardware corrector to produce a balanced distribution of "0" and "1" bits.
- A von Neumann corrector converts pairs of bits into output bits by converting the bit pair [0,1] into an output 1, converting [1,0] into an output 0, and outputting nothing for [0,0] or [1,1].
- The corrector prevents imbalances in the fast clock's duty cycle from biasing the output.

Most of random number generators work the same way

- A hardware random number generator typically consists of:
  - A transducer to convert some aspect of the physical phenomena to an electrical signal
  - An amplifier and other electronic circuitry to increase the amplitude of the random fluctuations to a macroscopic level,
  - Some type of analog to digital converter to convert the output into a digital number, often a simple binary digit 0 or 1.
- By repeatedly sampling the randomly varying signal, a series of random numbers is obtained.

Problems with hardware generated random numbers

- They are relatively slow
- They produce a limited number of random bits per second
- In order to increase the data rate, they are often used to generate the "seed" for a faster software generated random numbers, which then generates the output sequence

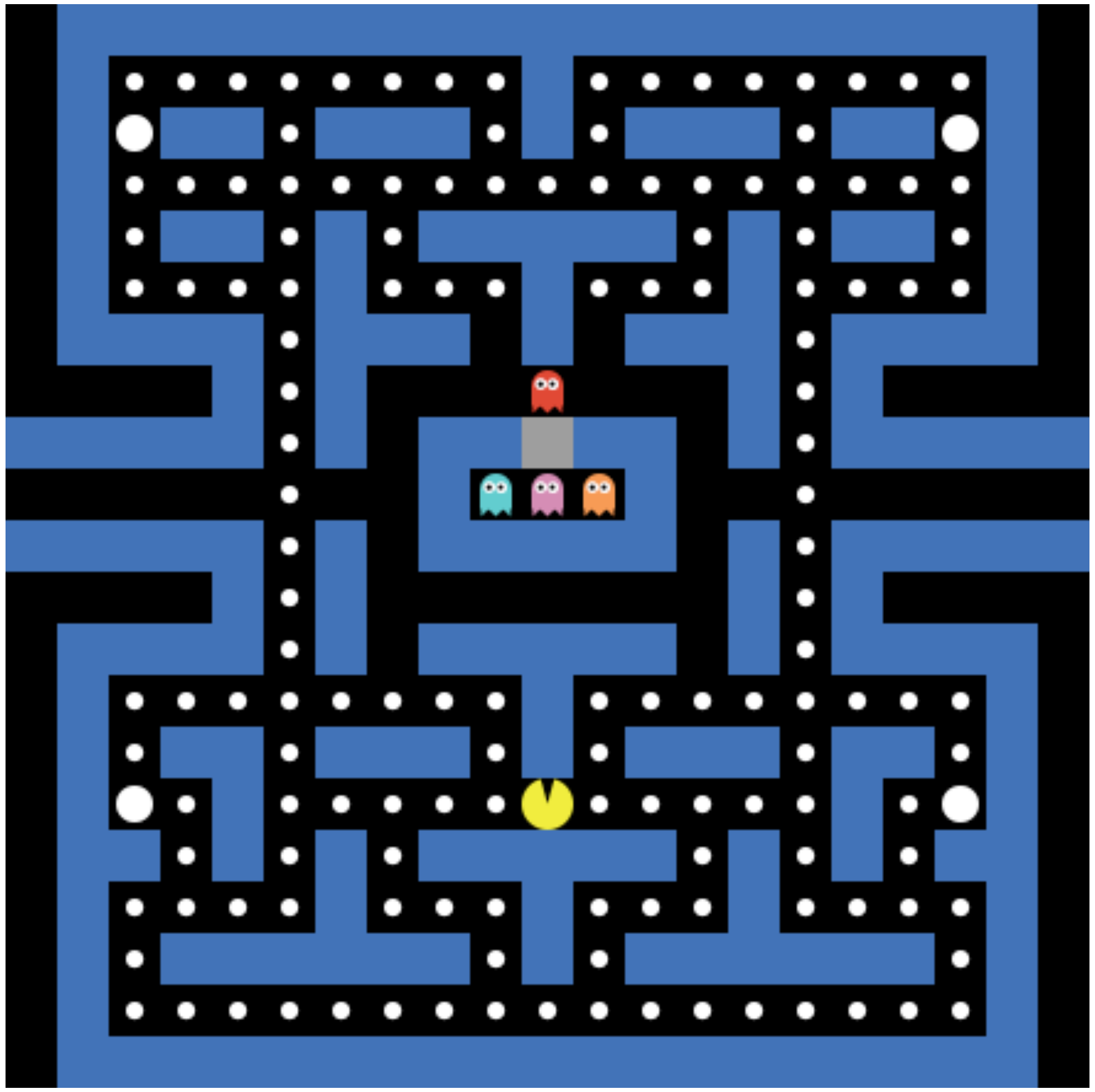
Algorithmically generated random numbers

Pseudo-random numbers generators (PRNG)

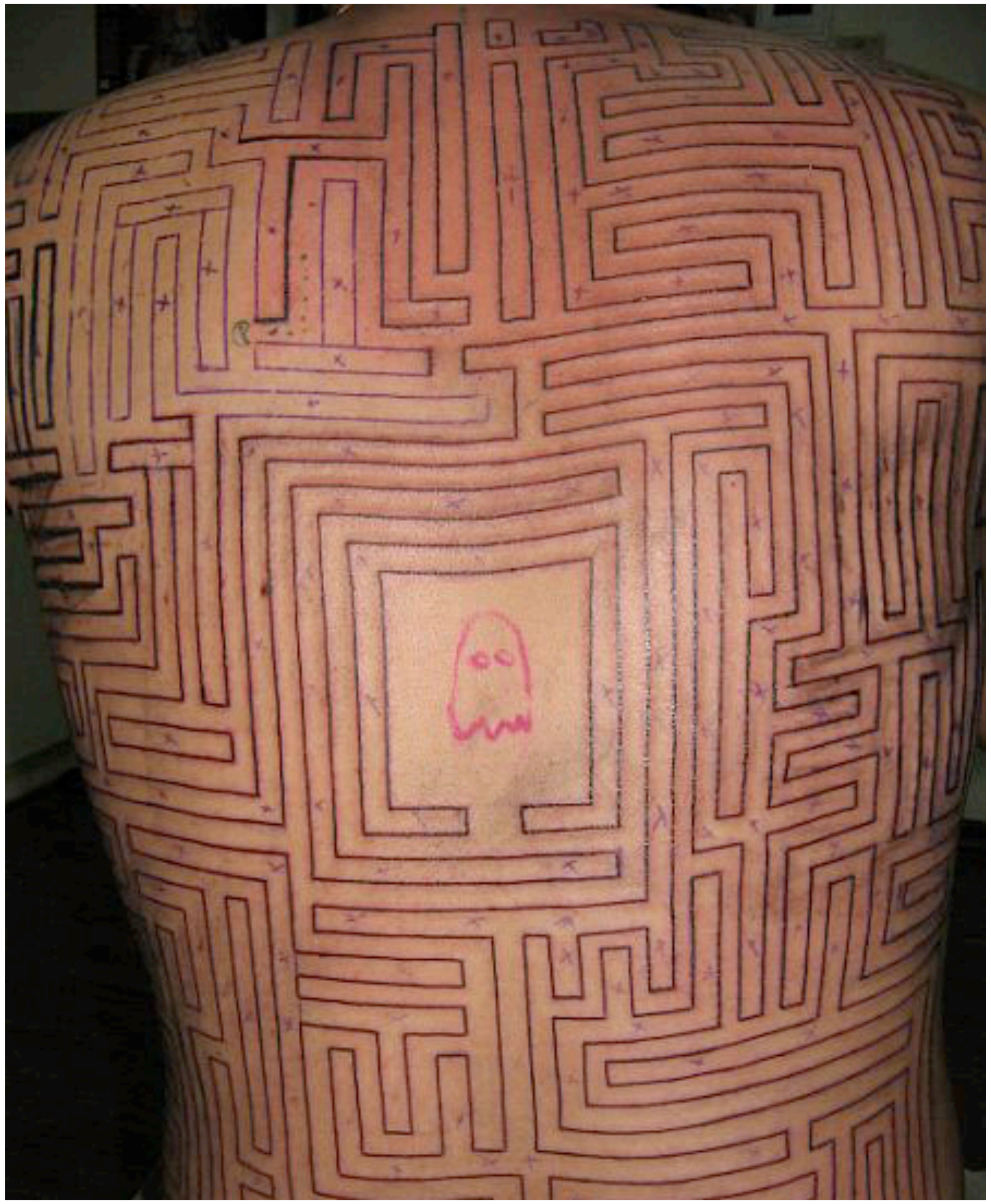
- Are software programs used in computers to generate "random" numbers.
- Use a deterministic algorithm to produce numerical sequences.
- Although sequences pass statistical pattern tests for randomness, by knowing the algorithm and the conditions used to initialize it, called the "seed", the output can be predicted.
- Because the sequence of numbers produced by a PRNG is predictable, data encrypted with pseudorandom numbers is potentially vulnerable to cryptanalysis.

Random Numbers vs Pseudo Random Numbers

- Hardware random number generators produce sequences of numbers that are not predictable, and therefore provide the greatest security.
- Pseudo-random numbers require a seed... so they are potentially unsafe.
- If I know the source code of the program, I can theoretically discover the seed of the random number... and predict the flow of a random process (such as pacman!)



Speaking of Pacman... This guy has a Pacman maze tattoo on his back.



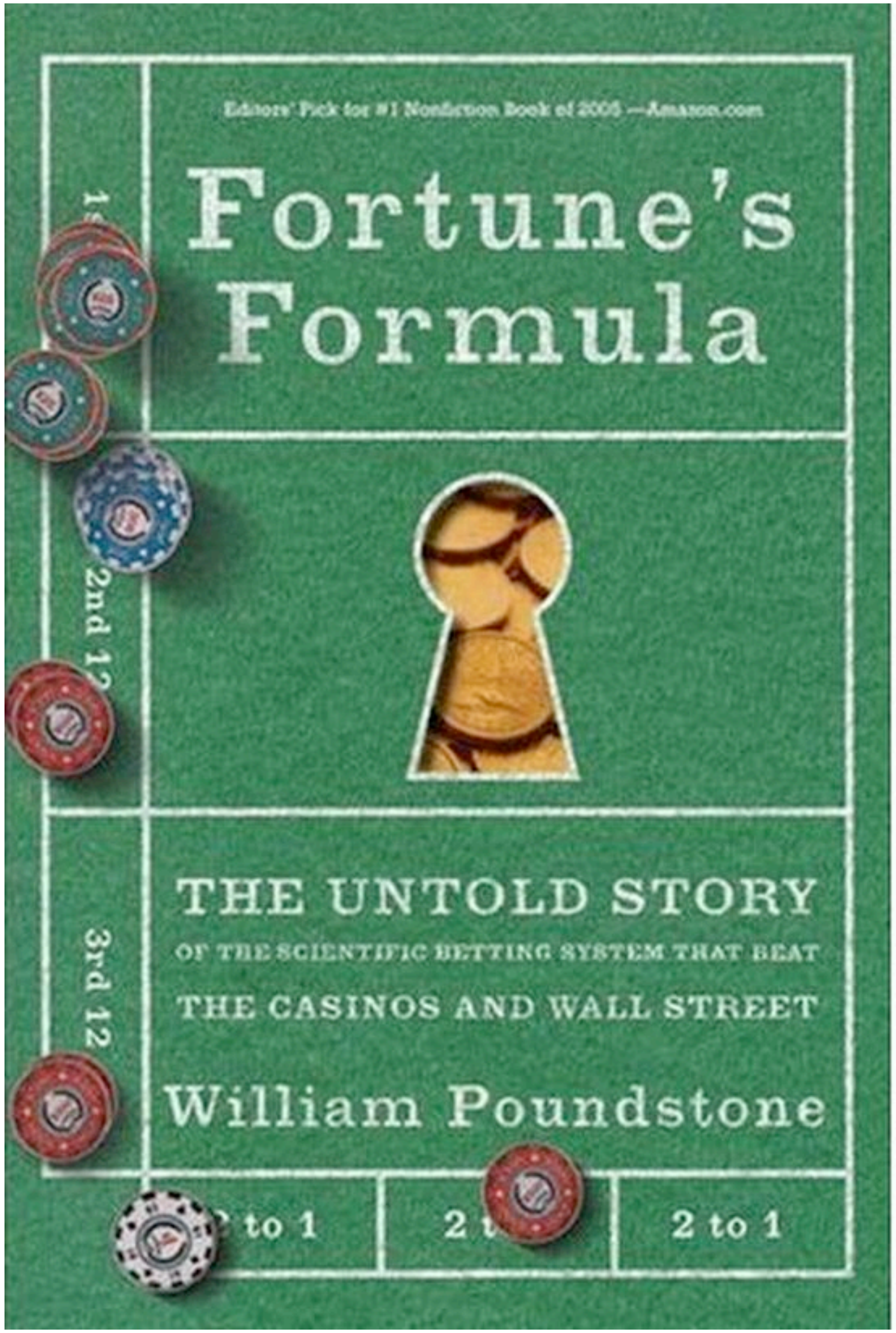
Random numbers by macroscopic processes

Numbers generated from "random" macroscopic processes

- Generated from activities such as coin flipping, dice, roulette wheels and lottery machines.
- These macroscopic processes are deterministic under Newtonian mechanics
- However the output of a well-designed device like a roulette wheel cannot be predicted in practice, because it depends on the sensitive, micro-details of the initial conditions of each use.

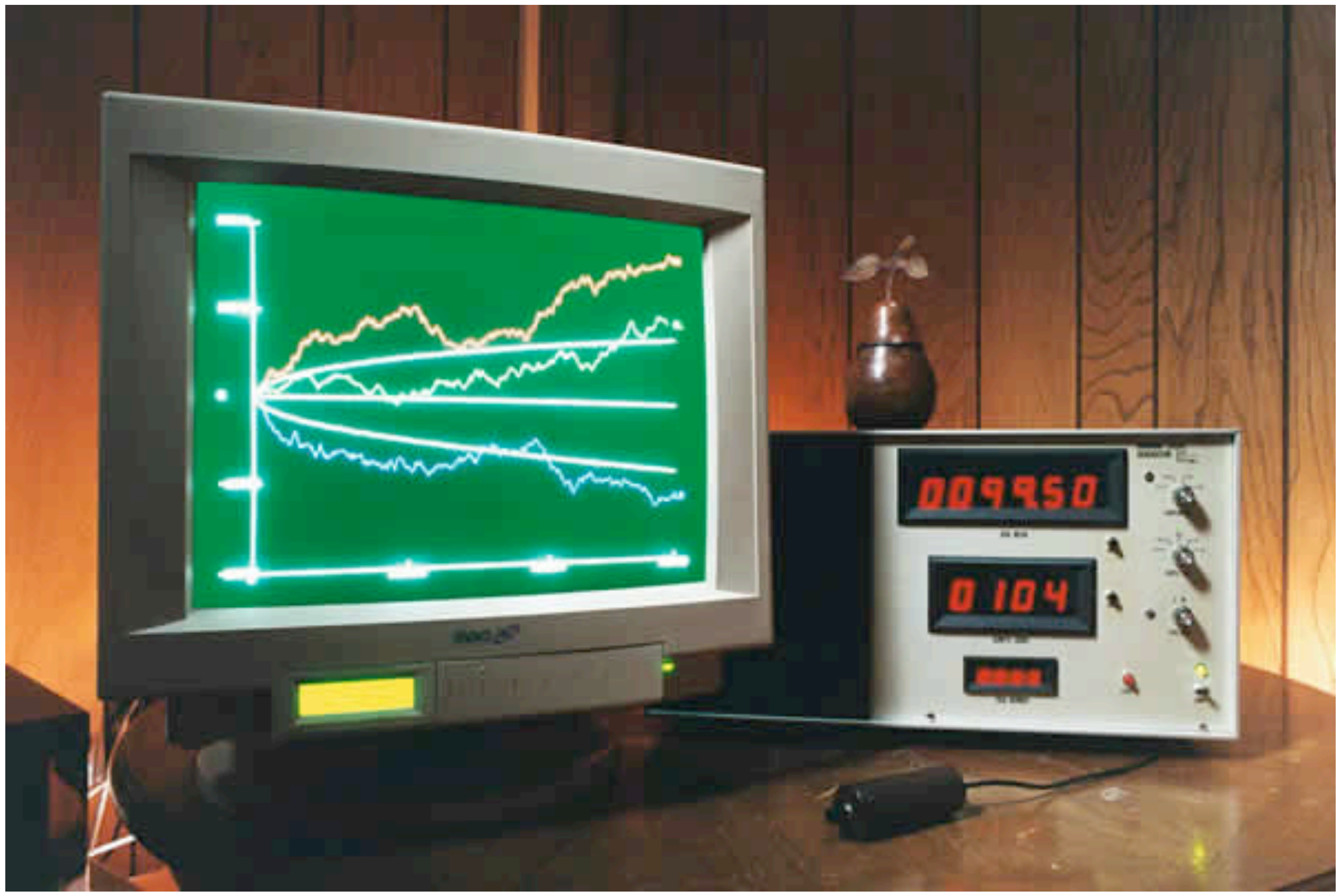
Edward Thorp (left) and Claude Shannon created the first wearable computer (1960) that was placed in a shoe that would suggest which octant to bet based on the speed of the roulette wheel.

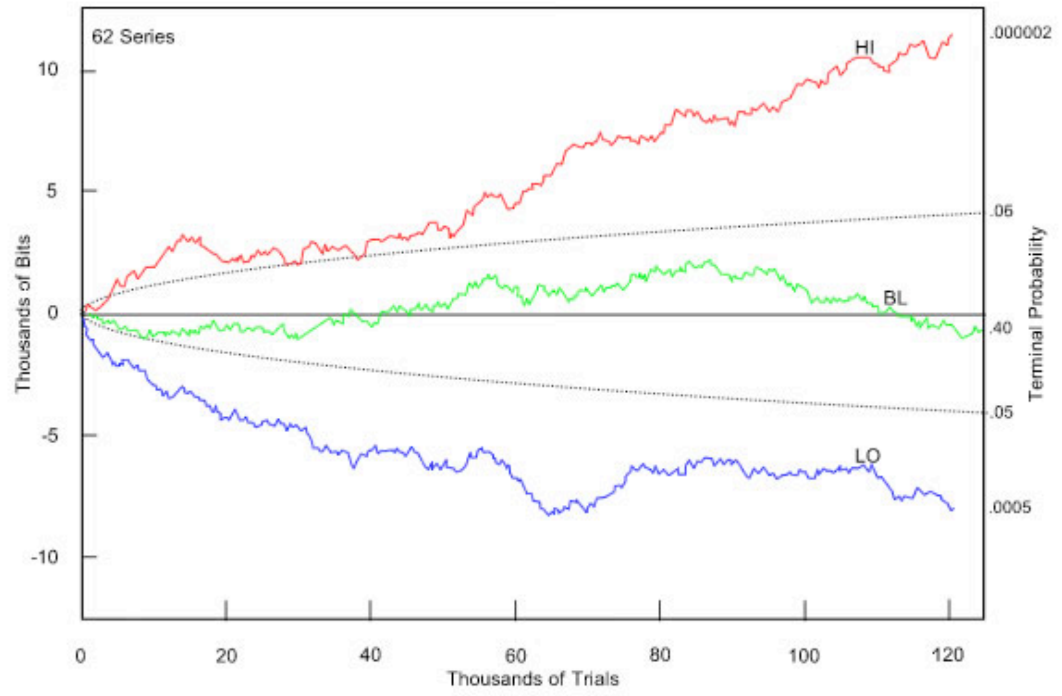
If you are interested in gambling, betting and random numbers I recommend reading this book!


Editors' Pick for #1 Nonfiction Book of 2005 —Amazon.com
Fortune's Formula
THE UNTOLD STORY OF THE SCIENTIFIC BETTING SYSTEM THAT BEAT THE CASINOS AND WALL STREET
William Poundstone
2nd 12
3rd 12
2 to 1

Parapsychology

- The Princeton Engineering Anomalies Research (PEAR) program was established at Princeton University in 1979.
- Their goal was to pursue rigorous scientific study of the interaction of human consciousness with physical devices.
- In other words, if I try really hard, can I mess up the outcome of a machine?





Cumulative deviations of REG/RNG (Random Event Generator/Random Number Generator) mean shifts achieved by conscious intention of one operator over some 375,000 experimental trials.

Can you turn on a green light?

If you try really hard, can you change the color of a traffic light with the power of your mind?

Can operators skew random distributions of mechanical processes?

http://www.princeton.edu/~pear/experiments.html

For different mechanical devices they published a lot of data since 1986.

Conclusions:

- Authors claim very small, but consistent across time and experimental designs, resulting in an overall statistical significance. On average, people can shift 2–3 events out of 10,000 from chance expectation!
- Opponents claim, only good data is being published (publication bias)

Closing of Princeton's PEAR lab

Dealing with skew of random numbers was not enough. They also did research on remote viewing.

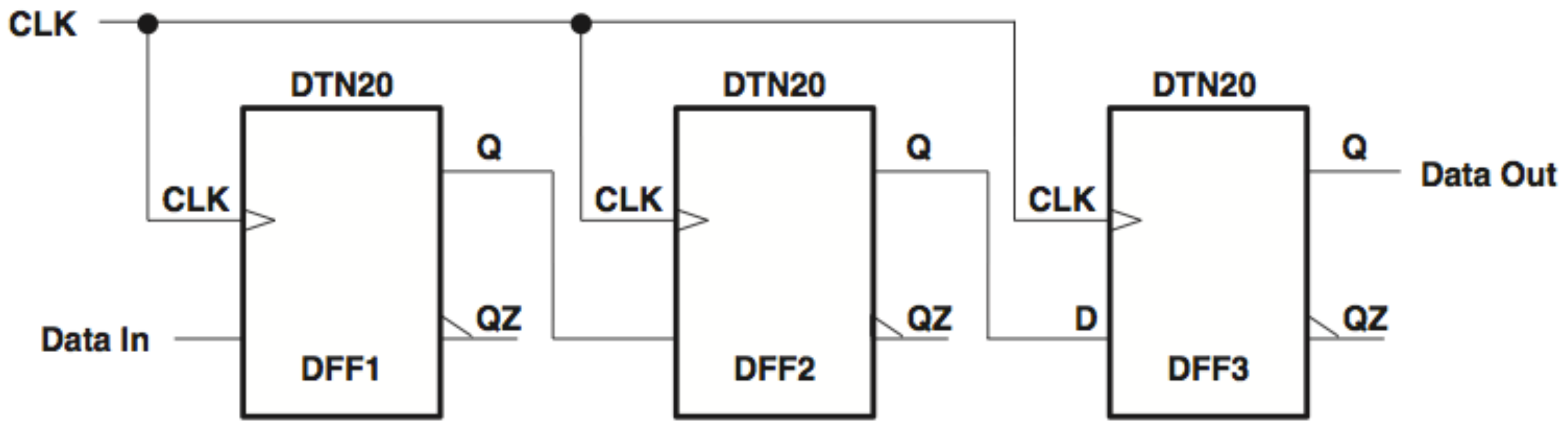
Remote viewing (RV) is the practice of seeking impressions about a distant or unseen target using paranormal means.

US government actually funded this type of research! http://www.remoteviewed.com

Robert L. Park said of PEAR, "It's been an embarrassment to science, and I think an embarrassment for Princeton".

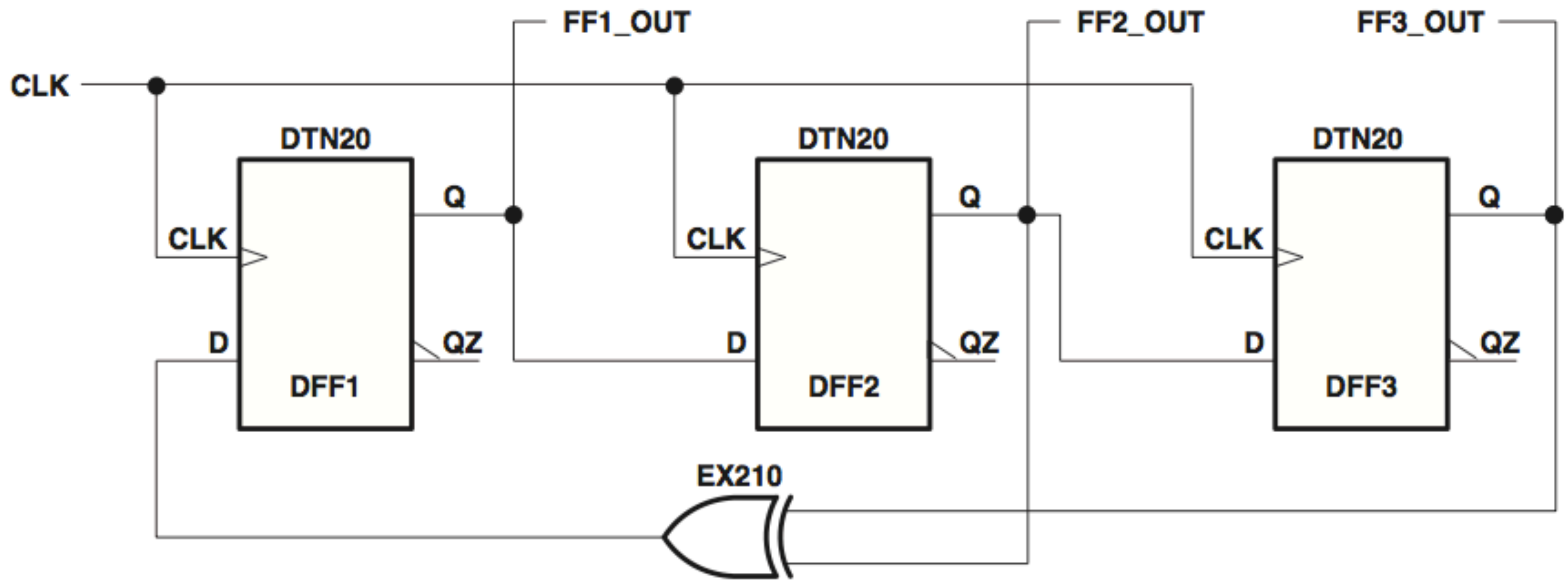
Pseudo random numbers in VHDL

We can create pseudo random numbers using a LFSR



A shift register when clocked, advances the signal through the register from one bit to the next most-significant bit.

Linear feedback shift register (LFSR)



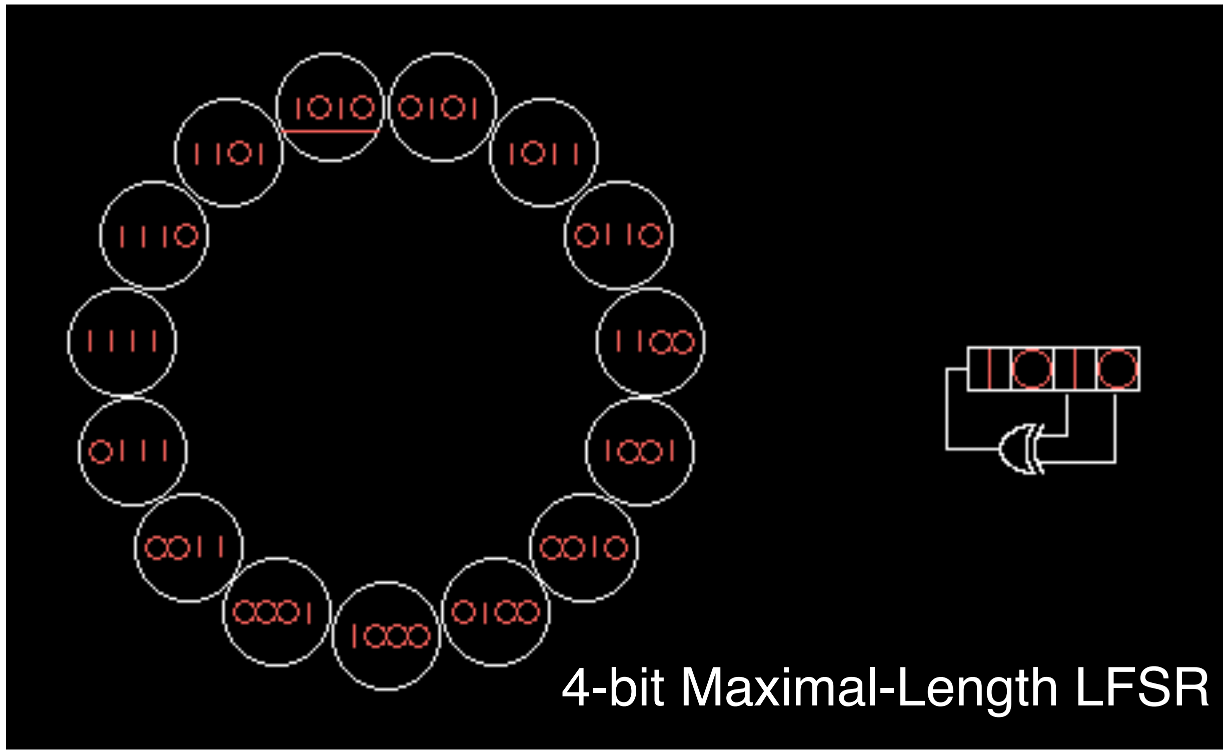
A linear feedback shift register can be formed by performing XOR on the outputs of two or more of the flip-flops together and feeding those outputs back into the input of one of the flip-flops.

Pseudorandom Pattern Generation

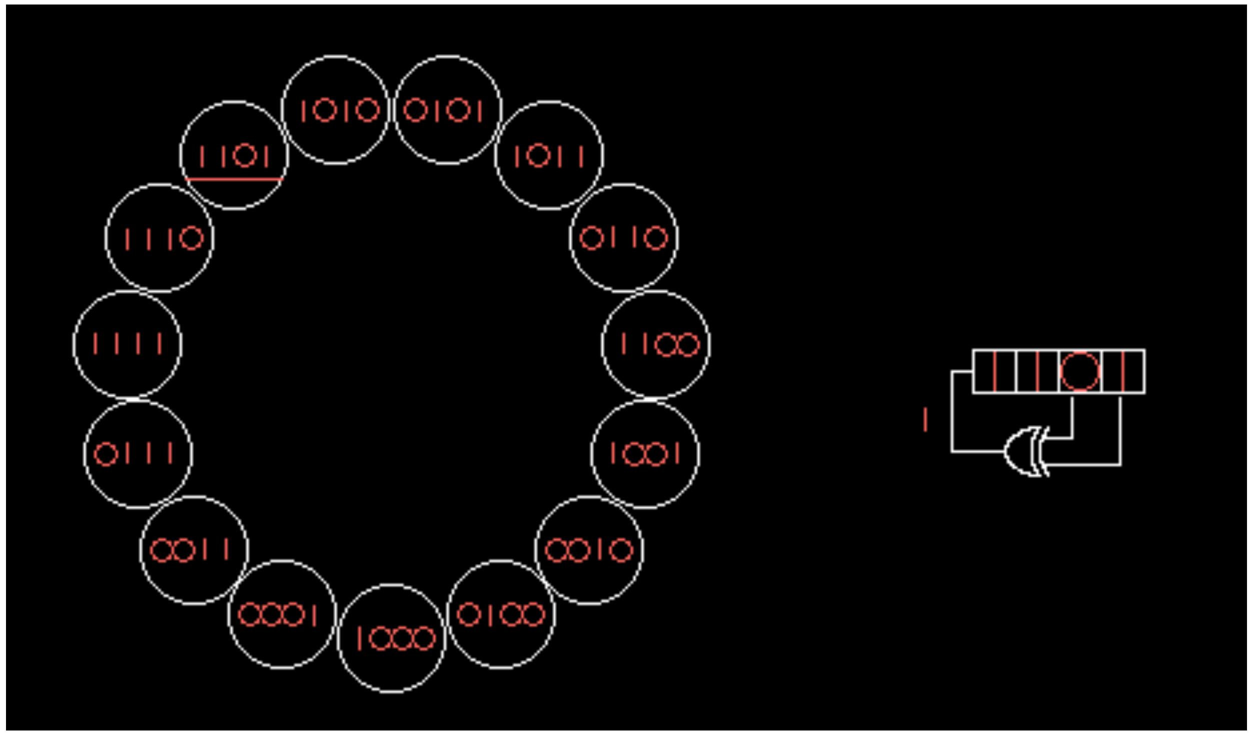
- Linear feedback shift registers make extremely good pseudorandom pattern generators.
- When the outputs of the flip-flops are loaded with a seed value (anything except all 0s, which would cause the LFSR to produce all 0 patterns) and when the LFSR is clocked, it will generate a pseudorandom pattern of 1s and 0s.
- Note that the only signal necessary to generate the test patterns is the clock.

Maximal-length LFSR

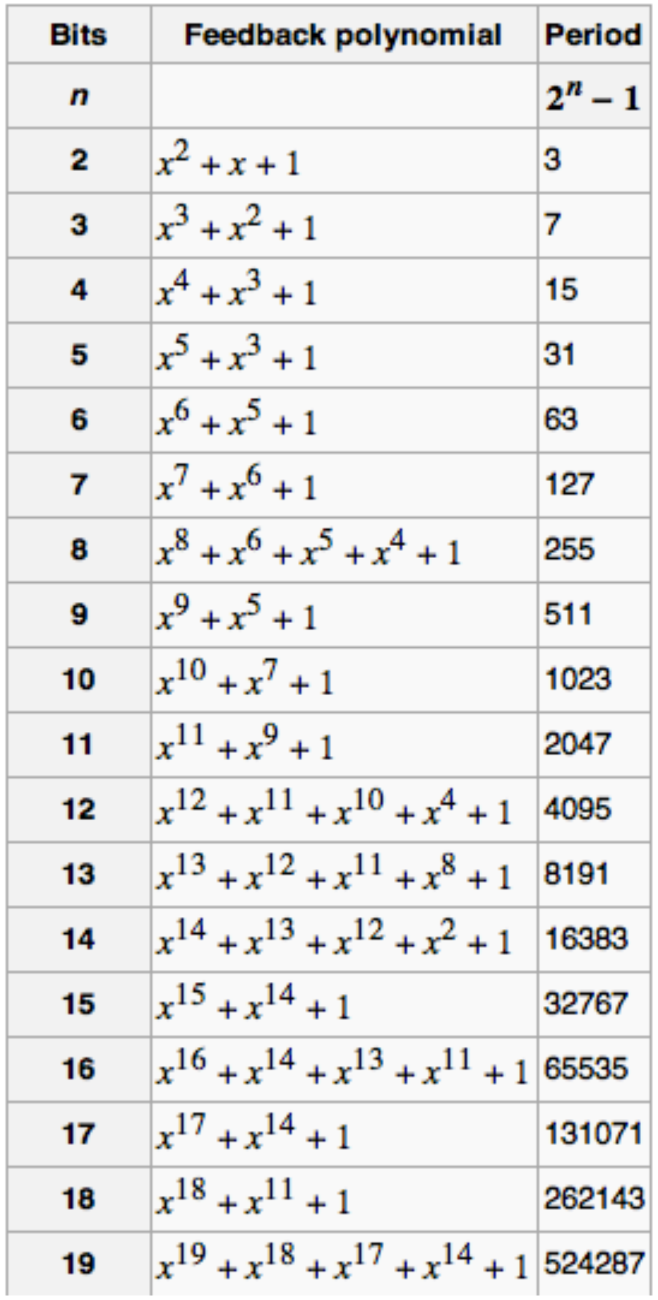
- A maximal-length LFSR produces the maximum number of PRPG patterns possible and has a pattern count equal to 2n – 1, where n is the number of register elements in the LFSR.
- It produces patterns that have an approximately equal number of 1s and 0s and have an equal number of runs of 1s and 0s.



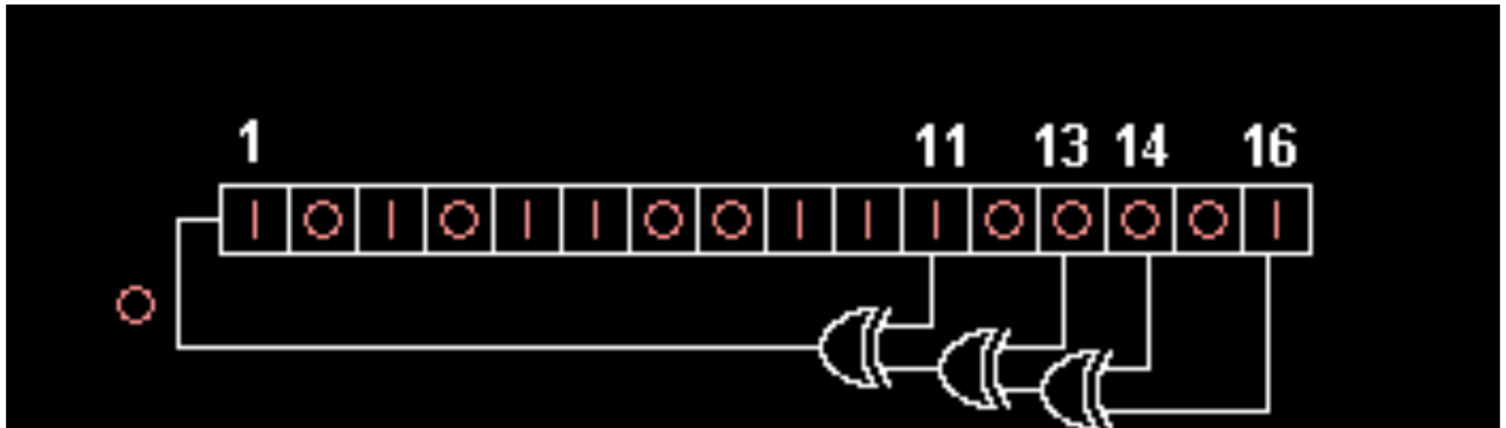
4-bit Maximal-Length LFSR



Maximal-length LFSR

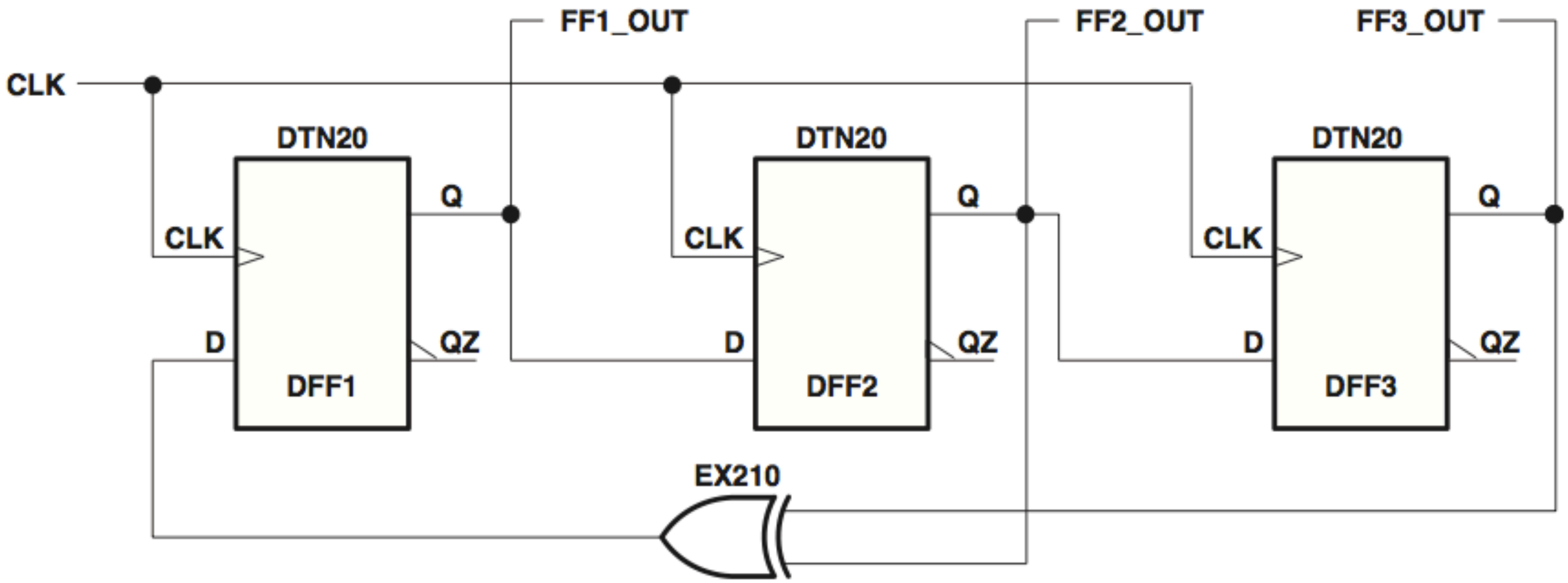


Bits	Feedback polynomial	Period
n		$2^n - 1$
2	$x^2 + x + 1$	3
3	$x^3 + x^2 + 1$	7
4	$x^4 + x^3 + 1$	15
5	$x^5 + x^3 + 1$	31
6	$x^6 + x^5 + 1$	63
7	$x^7 + x^6 + 1$	127
8	$x^8 + x^6 + x^5 + x^4 + 1$	255
9	$x^9 + x^5 + 1$	511
10	$x^{10} + x^7 + 1$	1023
11	$x^{11} + x^9 + 1$	2047
12	$x^{12} + x^{11} + x^{10} + x^4 + 1$	4095
13	$x^{13} + x^{12} + x^{11} + x^8 + 1$	8191
14	$x^{14} + x^{13} + x^{12} + x^2 + 1$	16383
15	$x^{15} + x^{14} + 1$	32767
16	$x^{16} + x^{14} + x^{13} + x^{11} + 1$	65535
17	$x^{17} + x^{14} + 1$	131071
18	$x^{18} + x^{11} + 1$	262143
19	$x^{19} + x^{18} + x^{17} + x^{14} + 1$	524287



16 bit Maximal LFSR

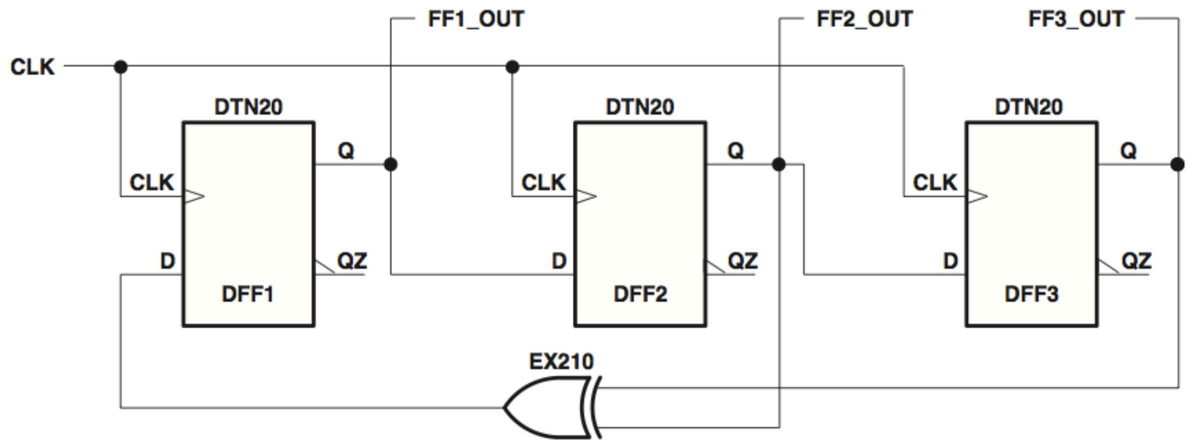
Because there is no way to predict mathematically if an LFSR will be maximal length, Peterson and Weldon2 have compiled tables of maximal-length LFSRs to which designers may refer.

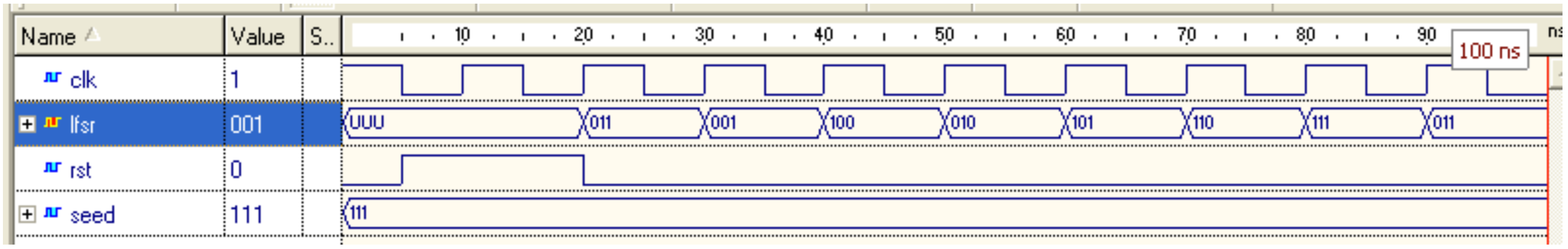


CLOCK PULSE	FF1_OUT	FF2_OUT	FF3_OUT	COMMENTS
1	1	1	1	Seed value
2	0	1	1	
3	0	0	1	
4	1	0	0	
5	0	1	0	
6	1	0	1	
7	1	1	0	
8	1	1	1	Starts repeat

Implementing a LFSR in VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity lfsr_block is
port (clk,rst : in std_logic;
seed : in std_logic_vector(2 downto 0);
lfsr : out std_logic_vector(2 downto 0)
);
end entity;

architecture myarch of lfsr_block is
begin
--signal lfsr : std_logic_vector(14 downto 0);
process(clk,rst)
variable tmp_lfsr : std_logic_vector(2 downto 0);
begin
    if (rst='1') then tmp_lfsr := seed;
    elsif (clk'event and clk='1') then
        tmp_lfsr :=(tmp_lfsr(1) xor tmp_lfsr(0)) & tmp_lfsr(2 downto 1);
        lfsr <= tmp_lfsr;
    end if;

end process;

end architecture;
```





Practical LFSR

- In a practical ASIC design, the user would create an LFSR that is much bigger than three bits to get a large number of pseudorandom patterns before the patterns repeated.
- However, there are some practical restrictions to the length of the LFSR.
- A 32-bit maximal-length LFSR would create over 4 billion patterns that, at a 16-MHz clock rate, would take almost 5 minutes to generate the whole pattern set.