LFSR based extractor for random number processing

Shi Yicheng

December 13, 2014

Overview

The LFSR(Linear Feedback Shift Register) based randomness extractor was originally designed as an algorithm for randomness extraction of a physical Random Number Generator(RNG).

Random numbers generated directly from a physical random number generator are usually biased and have inter-bits correlations, and are not considered useful for most cryptographic purposes. A randomness extractor receives raw random bits from a physical random number generator and outputs a random stream that is uncorrelated and bias free, converting a weak randomness stream into a strong one to suite most applications.

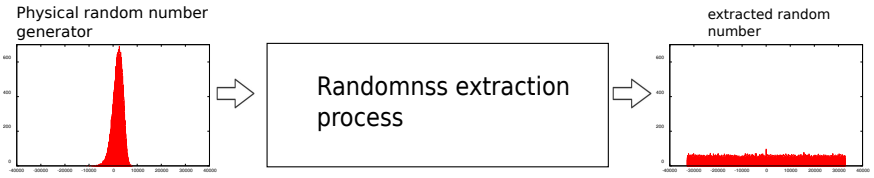

Figure 1: randomness extraction process. The input stream comes from a physical random number generator, and follows a Gaussian distribution(left). The extractor takes in the stream and produce an output stream that follows a uniform distribution(right).

Real-life implementations of randomness extractor do appear in various literatures. Most common methods involves using different cryptographic hashing functions. These methods are in general proven to be effective and secure, but the complexity of the hashing functions slows down the processing speed and increases the implementation difficulty.

We propose here a new randomness extractor based on a LFSR structure. The construct is compact and can be expected to operate at ultra-fast frequency.

Basic Structure

Fig 2 shows the basic structure of the LFSR based extractor. The central part is a Fibonacci type LFSR, with the LFSR taps chosen to give a maximal period. The extractor has a serial input and output. On each clock cycle, the following

operations take place:
1. One bit is read from the physical RNG to input.
2. The input bit is XORed with the bits from the LFSR taps, generating one new bit. The operation can be expressed as: $X_{output} = X_1 \oplus X_2 \oplus X_{input}$
3. The LFSR shifts, all bits move one position up, the vacated bottom position is occupied by the newly generated bit. The new bit is at the same time sent to output.
4. A portion of bits are dropped to conserve entropy.

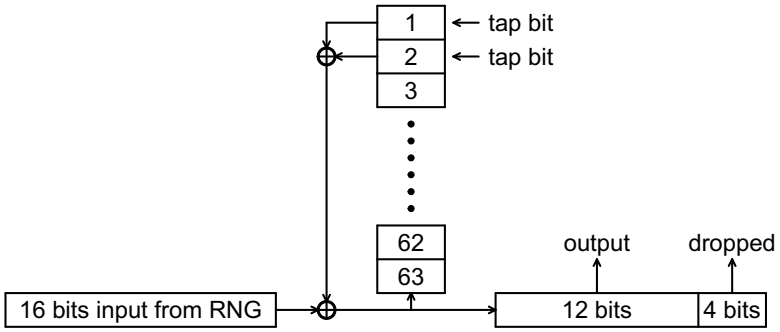

Figure 2: Example of a 63-bit, serial-in/out randomness extractor. The input random numbers are grouped in 16-bit words. Addresses 1 and 2 are the LFSR taps.

In this 63-bit structure, we intentionally add a bit dropping step to conserve the entropy of the input and output. A statistical measurement of the input yields 12 bits of Shannon entropy within every 16 bits. In response, for every 16 bits output, we drop 4 bits such that the process becomes irreversible.

The exact length of the LFSR is subject to different applications. A longer LFSR will be more suitable if security issues are of concern; shorter LFSR on the other hand will make the whole processing unit more compact.

Parallel-Structure Extractor

The basic structure shown in the last section has a serial input/output. In order to enhance the performance we propose a parallel structure extractor based on the same principle.

Fig 3 shows an example of a 63-bit parallel extractor. The addresses of the shift register are labelled in numbers. The arrows indicate the shift of bits at each clock cycle. To generate an output of 63 bits in parallel, $63 \times 2 = 126$ cells of memory are needed. Cells $1 \sim 63$ store the state of the LFSR, and are to be sent to output at next clock cycle; cells $64 \sim 126$ are used to store the next 63 bits that are newly generated by the XOR(Exclusive OR) module.

Similar to the serial extractor, the operations in one clock cycle are as fol-

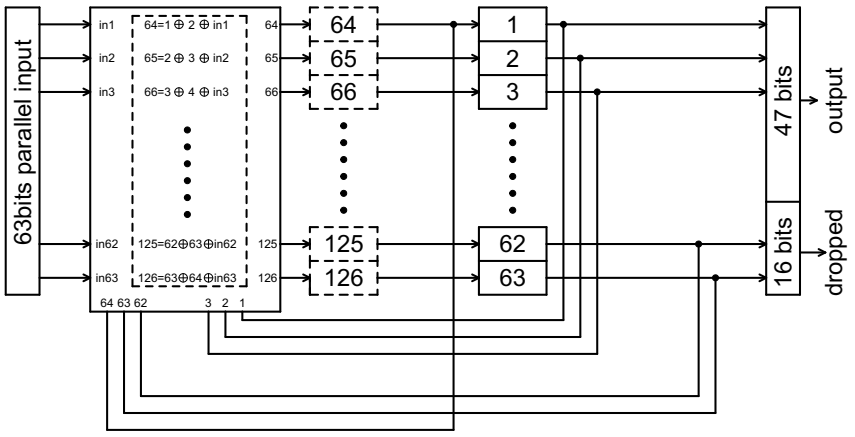

Figure 3: 63 bits parallel extractor. The input is grouped in 63 bits words.

lowing:
1. 63 bits are read from the physical RNG to the input.
2. XOR operations are performed between the input 63 bits and the bits stored in the shift register. Following the truth table below:

$$
\begin{aligned}
X_{64} &= X_1 \oplus X_2 \oplus X_{input1}, \\
X_{63} &= X_2 \oplus X_3 \oplus X_{input2}, \\
&\vdots \\
X_{126} &= X_{63} \oplus X_{64} \oplus X_{input63}
\end{aligned}
$$

this step will generate 63 new bits. It's worth noting here that all these XOR operations are performed in parallel, thus no extra operating time is required compare to the serial extractor.
3. The parallel LFSR shifts. All bits shift 63 addresses forward: $64 \rightarrow 1$, $65 \rightarrow 2$, etc. Bits originally stored in addresses $1 \sim 63$ are sent to output; the vacated addresses $64 \sim 126$ accommodate the newly generated 63 bits from the XOR operations.
4. A portion of bits are dropped to conserve entropy.

The parallel structured extractor will speed up processing speed at a cost of requiring more memory space and more XOR gates compare to its serial version. To build a parallel extractor using a n-bit LFSR that has k bit parallel output, k must be less or equal to n. The number of memory cells need is simply $n + k$.

performance

As a preliminary assessment, the 63-bit parallel extractor was used to process raw bits generated from a random number generator. The output unbiased

stream is sent through a randomness test suite to test for validity. The processed stream passed the randomness test: no obvious bias and correlations are detected in the output stream.