LFSR based scrambler for random number processing

December 8, 2014

This LFSR(Linear Feedback Shift Register) based randomness extractor was originally designed as a post-processing algorithm for randomness extraction purpose. Its input receives raw random bits from a physical random number generator(RNG) and outputs a random stream that is uncorrelated and bias free. The construct is compact and can be expected to operate at ultra-fast frequency.

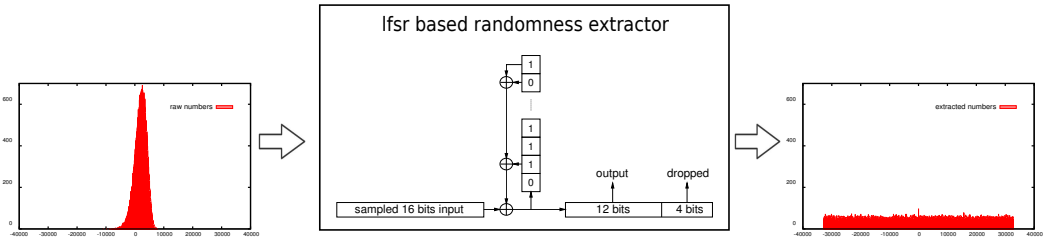

Figure 1: randomness extraction process. The input stream comes from a physical random number generator, and follows a Gaussian distribution(left). The extractor takes in the stream and produce an output stream that follows a uniform distribution(right).

Fig 2 shows the very basic structure of the extractor. The central part is a Fibonacci type LFSR. In this configuration, the bits from the LFSR taps are XORed with the incoming random bit to generate one new bit. The new bit is simultaneously sent to output and also back to the LFSR memory.

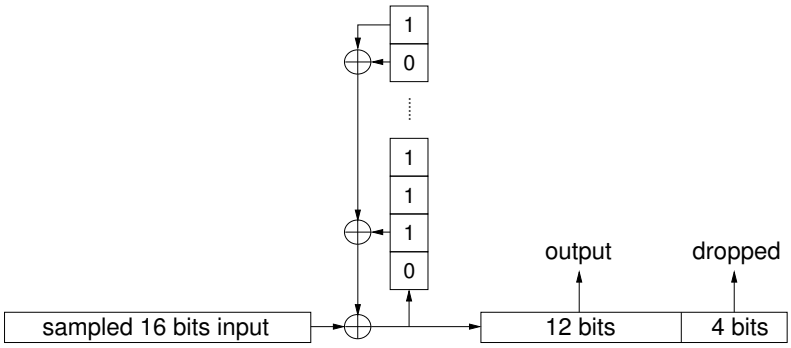

Figure 2: Basic structure of a 16-bit, serial-in/out randomness extractor

For each clock cycle, the extractor generates one bit output and update the internal state of the LFSR. In this 16-bit structure, we intentionally add a bit dropping step to ensure that the output entropy is no more than what's measured from the input numbers.

parallel-structure extractor

Here we also propose a parallel-structured extractor that has the same functionality.

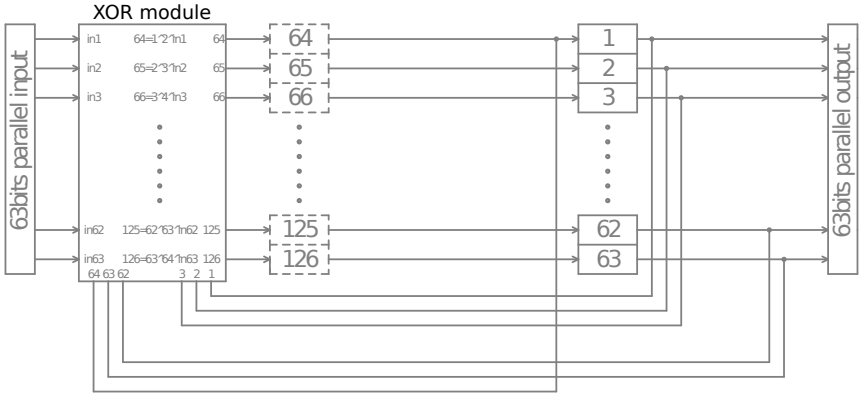

Figure 3: 63 bits parallel extractor.

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.

On each cycle, the extractor takes in 63 bits data generated by the RNG. The bits stored in addresses $1 \sim 64$ will be read into the XOR module and XORed with the 63 input random bits. The truth table for the XOR module is shown in the figure, which can be interpreted as:

$$\begin{aligned} X_{64} &= X_1 \oplus X_2 \oplus X_{in1}, \\ X_{63} &= X_2 \oplus X_3 \oplus X_{in2}, \\ &\vdots \\ X_{126} &= X_{63} \oplus X_{64} \oplus X_{in63} \end{aligned}$$

Once the XOR operations are done, all bits in the shift register shift one step to right: bits $1 \sim 63$ go to the parallel output, bits $64 \sim 126$ occupies addresses $1 \sim 63$, and addresses $64 \sim 126$ accommodates the 63 newly generated bits from XOR module.