Arithmetic Operators for On-The-Fly Evaluation of TRNGs

Renaud Santoro[a,b], Arnaud Tisserand[a], Olivier Sentieys[a], and Sébastien Roy[b]

[a] IRISA, INRIA Centre Rennes - Bretagne Atlantique, CNRS, Univ. Rennes 1
6 rue Kérampont, F-22305 Lannion, France.

[b] Département de Génie Electrique et de Génie Informatique
Université Laval, Québec, Qc, G1K7P4, Canada

ABSTRACT

Many cryptosystems embed a high-quality true random number generator (TRNG). The randomness quality of a TRNG output stream depends on its implementation and may vary due to various changes in the environment such as power supply, temperature, electromagnetic interferences. Attacking TRNGs may be a good solution to decrease the security of a cryptosystem leading to lower security keys or bad padding values for instance. In order to protect TRNGs, on-the-fly evaluation of their randomness quality must be integrated on the chip. In this paper, we present some preliminary results of the FPGA implementation of functional units dedicated to statistical tests for on-the-fly randomness evaluation. In the entropy test the evaluation of the harmonic series at some ranks is required. Usually its approximation is costly. We propose a multiple interval polynomial approximation. The decomposition of the whole domain into small sub-intervals leads to a good trade-off between the degree of the polynomial (i.e. multipliers cost) and the memory resources required to store the coefficients for all sub-intervals.

Keywords: true random number generator, randomness evaluation, statistical test, entropy test, harmonic series approximation, computer arithmetic operator, polynomial approximation, FPGA

1. INTRODUCTION

A random number generator (RNG) produces a random stream of bits which should be statistically independent, uniformly distributed and unpredictable. RNGs are necessary in many applications such as cryptography, communication, VLSI testing and probabilistic algorithms. There are two basic kinds of RNG: true random number generator (TRNG) and pseudo random number generator (PRNG). TRNGs are based on a physical noise source (e.g. radioactive decay, thermal noise or free running jitter oscillators). They usually have a small throughput which is not sufficient for most of applications. Furthermore, their randomness quality strongly depends on the implementation and some environment parameters variations (power supply, temperature, electromagnetic interferences...). PRNGs are based on fast but deterministic algorithms. They lead to high throughput but they are periodic. So they must be (re)initialized by a TRNG. Pairing up a PRNG and TRNG into a hybrid RNG leads to high throughput and high quality random streams.

RNG randomness evaluation is usually performed by using a battery of statistical tests such as Diehard[1] and NIST.[2] The context of our work is the design of TRNGs on FPGAs.[3,4] In statistical tests, counting the number of bits or bit patterns is a simple but very frequent operation. In some tests, the evaluation of more complex operations is required. For instance, in the entropy test,[5] the evaluation of the harmonic series at rank i, $H_i = \sum_{k=1}^{i} \frac{1}{k}$, is required. Getting an approximation to the harmonic series is usually costly. In this

Further author information:
Renaud Santoro: E-mail: renaud.santoro@irisa.fr
Arnaud Tisserand: E-mail: arnaud.tisserand@irisa.fr
IRISA: Institut de recherche en informatique et systèmes aléatoires.
INRIA: Institut national de recherche en informatique et automatique.
CNRS: Centre national de la recherche scientifique.

Mathematics for Signal and Information Processing, edited by Mark S. Schmalz, Gerhard X. Ritter, Junior Barrera, Jaakko T. Astola, Franklin T. Luk, Proc. of SPIE Vol. 7444, 74440S · © 2009 SPIE
CCC code: 0277-786X/09/$18 · doi: 10.1117/12.826336

paper, we propose a method that produces multiple interval polynomial approximations well-suited for high-performance hardware implementations. High-degree polynomials lead to accurate but costly approximations. The evaluation of a degree d polynomial using the Horner scheme requires d multiplications and d additions. Lower degree polynomials only lead to good accuracy on small intervals. There is a trade-off between the number of sub-intervals and the efficiency (speed and area) of the operators. The method proposed below performs an optimized decomposition of the whole domain into several sub-intervals. On each sub-interval, only low-degree polynomials are required to achieve good accuracy.

Section 2 recalls standard methods devoted to the evaluation of the randomness quality in TRNGs. Hardware arithmetic operators solutions for function approximation are recalled in Section 3. Our solution, and its implementation on FPGAs, for the statistical tests and especially the entropy test based on the evaluation of the harmonic series at some ranks are described in Section 4.

2. TRNG EVALUATION

2.1. Statistical Tests

In this section, the objective is to discuss the TRNG randomness as a function of the generator data rate, the FPGA circuit and the activity surrounding the TRNG. In practice, TRNG randomness evaluation is performed by using statistical tests. Statistical tests are implemented using high-level software programming. When an RNG is evaluated, a huge bit stream is stored in memory and then submitted to software tests. These batteries are specifically designed for PRNGs. TRNG validation is more complicated because a generator's behavior depends on its construction, on the external environment and essentially on a physical noise source which can differ in practice from an ideal noise. In[6] a methodology to evaluate physical generators is described. The procedure is based on TRNG construction and is the technical reference for AIS 31[7] which is the German evaluation and certification scheme for TRNG validation. Another standard can be used to estimate the TRNG randomness,[8] namely FIPS 140-1/2.[9]

The large number of statistical tests and the bit stream size required for each test makes software processing time consuming. As presented in Figure 1, the proposed methodology consists in evaluating the TRNG quality by using some hardware accelerated statistical tests. This avoids the interfacing problem with an external operating system, as only the test results must be transferred. However, the hardware tests must be very efficient to be implemented into embedded circuits and to allow the TRNG analysis to proceed in real time. Moreover, the area occupied by the tests must be reduced and the test must be placed in the FPGA far enough from the TRNGs to avoid disrupting their behaviors. Statistical tests described in FIPS 140-2 and in AIS 31 have been selected as the randomness metric. The FIPS 140-2 battery is comprised of four tests: the *frequency* test, the *poker* test, the *run* test and the *long-run* test. These tests analyze a 2×10^4-bit stream. The tests provide explicit bounds that the computed results must satisfy. In AIS 31, TRNGs are divided into two functionality classes, P1 and P2. A TRNG is said to belong to P1[7] if the internal random numbers pass a certain number of statistical tests described in.[6] P1-TRNGs are not sufficiently secure for applications such as session key generation, signature key pairs or signature parameters.[10] Consequently, the more restrictive P2 class has been introduced. In P2-TRNGs, the digitized analog signal (DAS) is also tested. To certify a P2-TRNG, the AIS 31 standard includes 9 statistical tests. The tests T0-T5 are applied to the internal random bits. Tests T1-T4 are the same tests as those used in the first FIPS 140-2 version, the FIPS 140-1. Then, the tests T6-T8 are applied to the DAS. Test T6 is a uniform distribution test which consists of two sub-tests. Test T7 is a comparative test for multinomial distributions and is also decomposed into two sub-tests. Finally, Test T8 is an entropy test introduced in[5] which estimates the source entropy. In this work, the FIPS 140-2 and the AIS 31 statistical tests excepted test T0 (due to memory restrictions) have been implemented into hardware.

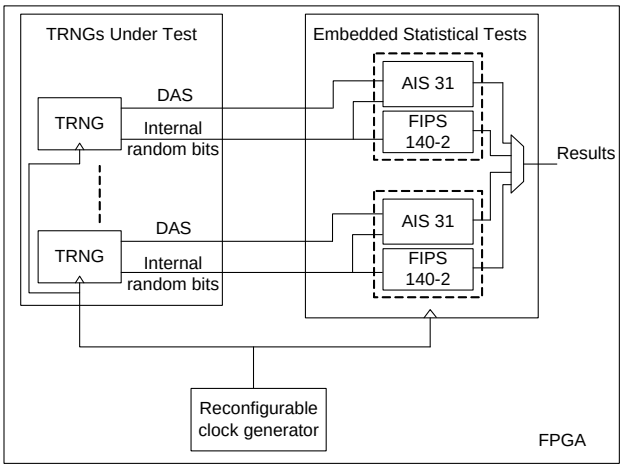

Figure 1. TRNG evaluation methodology.

2.2. Entropy Test

The entropy test[5] is based on Maurer's Universal Test (TUM).[11] This test is included into statistical test batteries such as NIST.[2] It allows detection of weaknesses into random sequences. It asymptotically measures the entropy of a random source. In,[12] a modified version of TUM was proposed with a more accurate result.

In the entropy test, the analyzed binary sequence $b_1, ..., b_{(Q+K)L}$ is segmented into $K+Q$ non-overlapping L-bit words, $w_1, ..., w_{Q+K}$. The test is based on the computation of the minimal distance (A_n) from w_n to the latest word having the same value:

$$A_n = \begin{cases} n \text{ if } \forall i < n, w_{n-i} \neq w_n \\ \min\{i | i \geq 1, w_n = w_{n-i}\} \text{ else.} \end{cases} \quad (1)$$

The result of

$$f_{T_U}^H(s^N) = \frac{1}{K} \sum_{n=Q+1}^{K+Q} g(A_n), \quad (2)$$

where

$$g(i) = \frac{1}{\log(2)} \sum_{k=1}^{i-1} \frac{1}{k} \quad (3)$$

is computed. As a function of the L, Q and K values, the value of (2) must lie in a specified interval. In AIS 31, the entropy test is used with the parameters: $L = 8$, $Q = 2560$ and $K = 256 \times 10^3$. According to,[6] if the result of (2) is greater than 7.976, the analyzed binary sequence satisfies the statistical test.

3. FUNCTION APPROXIMATION IN HARDWARE

Many algorithms and implementations can be found for basic arithmetic operators such as addition/subtraction or multiplication, for instance see.[13] For function evaluation the problem is more complex. Function approximation is often performed using polynomial evaluation in software as well as in hardware implementations. For instance, elementary functions (sine, cosine, exponential, logarithm, arc-tangent...) are often evaluated using polynomials (see[14]). In some digital signal processing applications, such as frequency demodulation, low degree polynomials are often used for evaluating reciprocals. Other algebraic functions, such as square root or square root reciprocal can be efficiently approximated using polynomials. The proposed multiple interval polynomial approximation is an extension to the polynomial approximation proposed in.[15–17]

The target function is f with input and output in fixed-point format (2's complement notation). The function f is approximated using the degree-d polynomial p. The argument x is in the domain $[a, b]$ and the result $p(x)$ is in the range $[a', b']$. Extension to other forms of input/output intervals such as $[a, b[$ is straightforward and is not considered here. Here, we consider function evaluation without range reduction (see[14]). The method below can be used in more complex schemes including range reduction steps. The argument x is a w_I-bit number and the output $p(x)$ is a w_O-bit number. The input argument x is considered as exact. The position of the binary point (i.e. the number of integer bits) is determined by the format of the smallest representation that include both a and b. The number of fractional bits will be computed by the generation method to fit the target accuracy requirements. The polynomial coefficients are denoted p_0, p_1, p_2, ..., p_d, then $p(x) = \sum_{i=0}^{d} p_i x^i$. The polynomial coefficients are represented using 2's complement.

Several evaluation schemes may be used to compute the value $p(x)$ in practice. In this work, we only consider the Horner scheme: $p(x) = p_0 + x(p_1 + x(p_2 + x(\cdots + xp_d)\cdots))$. The Horner scheme is implemented using basic blocks called *fused multiply and adds* (FMAs). An FMA computes $uv + w$ with more or less the same cost (delay and circuit area) than a single multiplication uv. The Horner scheme requires d additions and d multiplications to evaluate a degree-d polynomial. We use a sequential implementation (with d iterations) of the Horner scheme.

The numerical quality of the approximation to f using the polynomial p deals with two components: the *approximation error* and the *round-off error*. The approximation error measures the distance between the mathematical function f and the function used for the approximation, here the polynomial p. The approximation error is also called the *method error*. In order to measure the theoretical approximation error ϵ_{app} due to the use of the polynomial p to approximate the function f on $[a, b]$, we use the distance:

$$\epsilon_{\text{app}} = ||f - p||_\infty = \max_{a \le x \le b} |f(x) - p(x)|. \tag{4}$$

Here, the value $p(x)$ is the mathematical value computed using an infinite precision. The approximation error ϵ_{app} is the smallest theoretical error that can be obtained using the polynomial p for approximating f. Due to the finite precision of the coefficients and intermediate computations during the practical evaluation of p, we will have to deal with larger errors. In this work, the value of ϵ_{app} is numerically estimated using the Maple `infnorm` command. We assume that the approximated functions are "smooth" enough to ensure that `infnorm`'s result is correct (see[14]). Furthermore, we will use overestimations of ϵ_{app} to ensure that the generated operators fit the target accuracy.

The polynomial approximations used in the following are based on the *minimax* polynomial approximation as a starting point. The degree-d minimax polynomial approximation to f on $[a, b]$ is the polynomial p^* that satisfies:

$$||f - p^*||_\infty = \min_{p \in \mathcal{P}_d} ||f - p||_\infty, \tag{5}$$

where $\mathcal{P}_d$ is the set of polynomials with real coefficients and degree at most d. Minimax approximations can be computed thanks to an algorithm due to Remes (see[18]). More details about minimax approximations may be found in.[14] In this work the minimax polynomials are numerically computed using the Maple `minimax` command.

The *round-off error* or *rounding error* due to the discrete nature of the intermediate and final values adds up to the approximation error. It is the difference between the calculated approximation of a number and its exact mathematical value. This error is small for one single operation, i.e. a fraction of the weight of the least significant bit (LSB). But during a sequence of operations, these small errors may accumulate and significantly degrade the accuracy of the final result. In order to limit the rounding error, we introduce g additional *guard bits* for the intermediate computations (i.e. they are done on words of $w_O + g$ bits). The final accuracy validation is performed using exhaustive tests (in our case, we only have to handle 2^{13} different input values).

In the following, errors are expressed directly or as equivalent accuracy. The accuracy is the number of correct or significant bits. The relation between the error ϵ and the accuracy μ is $\mu = -\log_2 |\epsilon|$. For instance, the error $\epsilon = 0.0000107$ is equivalent to an accuracy of $\mu = 16.5$ correct or significant bits.

4. PROPOSED SOLUTION

4.1. Entropy Test Implementation

The entropy test implementation can be decomposed into two parts. In the first one, a 256×18-bit memory is required to save the index (n) of cycle in which the L-bit word (w_n) is encountered. Then, each time a new word is received, the minimal distance (A_n) is computed from the memory content. The $g(A_n)$ result is calculated and added to the summation (2).

In order to reduce the computation complexity of equation (2), only

$$K \log(2) f_{T_U}^H(s^N) \tag{6}$$

can be calculated. However, if the complexity is reduced, an efficient solution to evaluate the function $g(i)$ must be found. The function $g(i+1)$ is the i-th harmonic number

$$H_i = \sum_{k=1}^{i} \frac{1}{k} \tag{7}$$

and must be computed for $i \in \{1, ..., 258559\}$. The proposed solution to implement the function $g(i+1)$ is to use an approximation to H_i (see[19] for approximations to H_i). In this work, the DeTemple-Wang approximation,[20] accurate enough on the i domain, given by

$$H^{'}(i) = \log\left(i + \frac{1}{2}\right) + \gamma + \frac{1}{24\left(i + \frac{1}{2}\right)^2 + \frac{21}{5}} \tag{8}$$

has been selected, where γ is the Euler constant. In order to simplify, γ is subtracted from (8) and is included in the final test result:

$$H^{''}(i) = H^{'}(i) - \gamma. \tag{9}$$

4.2. Polynomial approximations to $H^{''}(i)$

The maximal error of (6) must be less than 1×10^{-3} (value from[7]). The accuracy must be at least equal to 9.96 correct bits. Equation 6 is obtained from 256.10^6 successive additions of (9). The maximal error for (9) is equal to $1/\left(256.10^6\right)$ which leads to a minimal accuracy of 27.93 correct bits.

In the first step, the objective is to find the minimax polynomials using Maple. The input i of $H^{''}(i)$ is an 18-bit integer. In order to speed-up the approximation, the input is normalized between $[2^{-18}, 258559 \times 2^{-18}]$. As a result, the function to approximate is redefined by

$$H^{'''}(i) = \log\left(i \times 2^{18} + \frac{1}{2}\right) + \gamma + \frac{1}{24\left(i \times 2^{18} + \frac{1}{2}\right)^2 + \frac{21}{5}}. \tag{10}$$

The function $H^{'''}(i)$ is presented in Figure 2. The function $H^{'''}(i)$ is approximated for $i \in [2^{-18}, 1[$.

4.3. Segmentation

The objective is to approximate the function $H^{'''}(i)$, with $i \in [2^{-18}, 1[$ and with at least 29.73-bit accuracy. Using only one polynomial for the whole interval is not possible in practice (its degree is too huge).

The domain of the function must be divided into segments and approximated on each segment. The most common approach is to use a uniform segmentation. The function is divided into equal segments. If the indexing problem to address the minimax polynomial coefficient is simple, the segments are not customized according

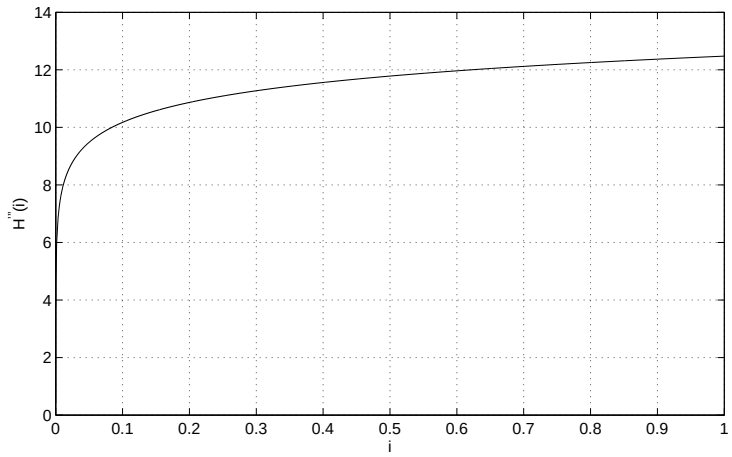

Figure 2. Approximated function $H^{'''}(i)$.

to the local function characteristics. As a result, depending on the function, the uniform segmentation can be inefficient. Another possibility is to use a nonuniform segmentation in which the segment widths can vary.

In[21] a segmentation scheme is proposed: $\Delta_i = \{US, P2S\}$ in which US and P2S are respectively the uniform segments and powers of two segments. It describes a two-level hierarchy segmentation $\Delta_0(\Delta_1)$ where Δ_0 and Δ_1 are respectively the outer and inner segments. The outer segments uses uniform segments or segments which vary as a power of two. The inner segmentation always employs a uniform segmentation. As a result, the hierarchy scheme proposed by[21] is $H = \{P2S(US), P2SL(US), P2SR(US), US(US)\}$. $P2SR$ (respectively $P2SL$) is a $P2S$ that decrease only to the left (respectively right) side. The number of bits required to address the outer segments (v_0) is a design constraint as well as the function to approximate, the maximal error and the hierarchy scheme H. Then, the number of inner segments to satisfy the accuracy constraint is searched. This step allows determination of the number of bits v_1 required to address the inner segments.

In this study, the nonuniform segmentation principle proposed by Lee et al. in[21] has been selected. As shown in Figure 2, the function is difficult to approximate for small values of i. As a result, an outer segmentation by a power of two that decreases only to the right side (called P2SR in[21]) has been selected. The principle of the P2SR is presented in Table 1 for an 8-bit input, $v_0 = 5$ and $v_1 = 3$. As a result, 5 outer segments and 8 inner segments can be addressed. Here, a different methodology than[21] to find v_1 has been realized. For a small v_0 value, the function non-linearity requires the use of a large number of inner segments or high degree minimax polynomials. In practice, implementing high-degree polynomials is costly. Moreover, using a large number of inner segments consumes a huge amount of memory. Consequently, here, the proposed methodology is to fix the degree-d of minimax polynomials to 3 and the number of bits for the outer segments to $v_0 = 18$ (the input width). Then, the number of inner segments needed to satisfy the minimal accuracy is searched.

For each outer segment, the algorithm starts with $v_1 = 0$ and searches the degree-3 minimax polynomial approximation $p^\star$ using Maple. If the obtained accuracy $-\log_2(\epsilon_{\text{app}})$ is less than 27.93 bits, then v_1 is incremented. The outer segment is decomposed in two inner segments. Then, the minimax polynomial approximation is computed and the error is analyzed. At each step, the number of inner segments is doubled. The algorithm is repeated until the 27.93 correct bits accuracy is reached.

Experiments show that each outer segment must be decomposed into 32 uniform segments, meaning that $v_1 = 5$. Figure 3 presents the outer segmentation for $i \in [2^{-18}, 1[$.

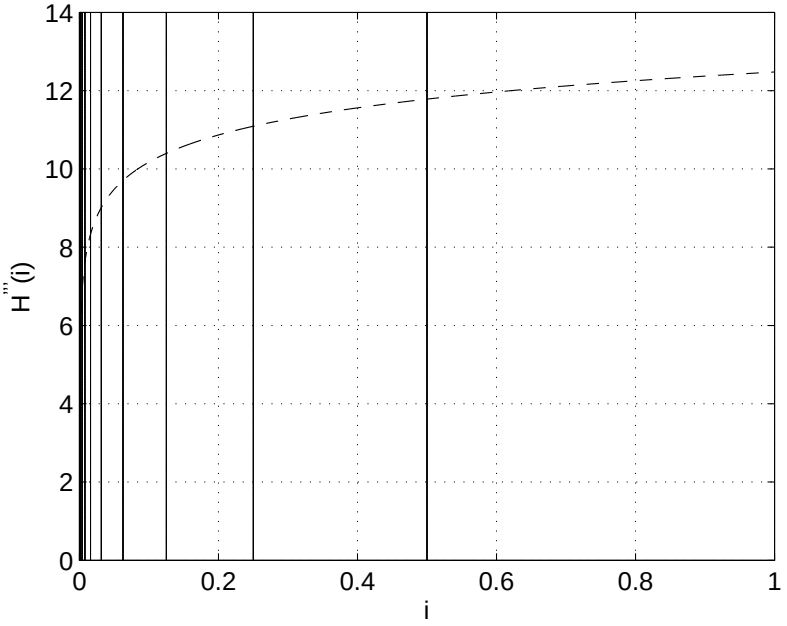

Figure 3. Outer segmentation of $H^{'''}(i)$ for $i \in [2^{-18}, 1[$.

P2SR address	range
0	00000|000 ⇒ 00000|111
1	00001|000 ⇒ 0000.|111
2	00010|000 ⇒ 0001.|111
3	00100|000 ⇒ 001..|111
4	01000|000 ⇒ 01...|111
5	10000|000 ⇒ 1....|111

Table 1. P2SR segmentation example for a 8-bit input, $v_0 = 5$ and $v_1 = 3$.

Then, the minimal number of bits required to represent the minimax polynomials in fixed-point for each inner segment is searched.[15] The fractional part to represent the polynomial coefficients is $n = 28$ bits, Figure 4 presents the approximation accuracy (in correct bits) when the polynomial coefficient fractional part is equal 28 bits.

As previously introduced, v_0 has been fixed to 18, implying that 18 outer segments decomposed into 32 inner segments are used. Some outer segments are not necessary. For example, the outer segment $[2^{-18}, 2^{-17}[$ is also divided into 32 uniform segments. As a result, the minimax approximation is more costly than a direct memorization of (10) for small values of i. In order to evaluate from which outer segment the approximation is useful, the efficiency of the polynomial approximation is evaluated in terms of memory size. Figure 5 shows the memory size (in bits) as a function of the method (approximation or direct memorization) to compute the function (10) for $i \in [2^{-18}, 2^{-18+j}[$. As presented, for $i \in [2^{-18}, 2^{-13}[$, the direct memorization is more efficient. Then, from $j = 6$, for $i \in [2^{-13}, 1[$, the minimax polynomial approximation is greatly less costly in term of memory units.

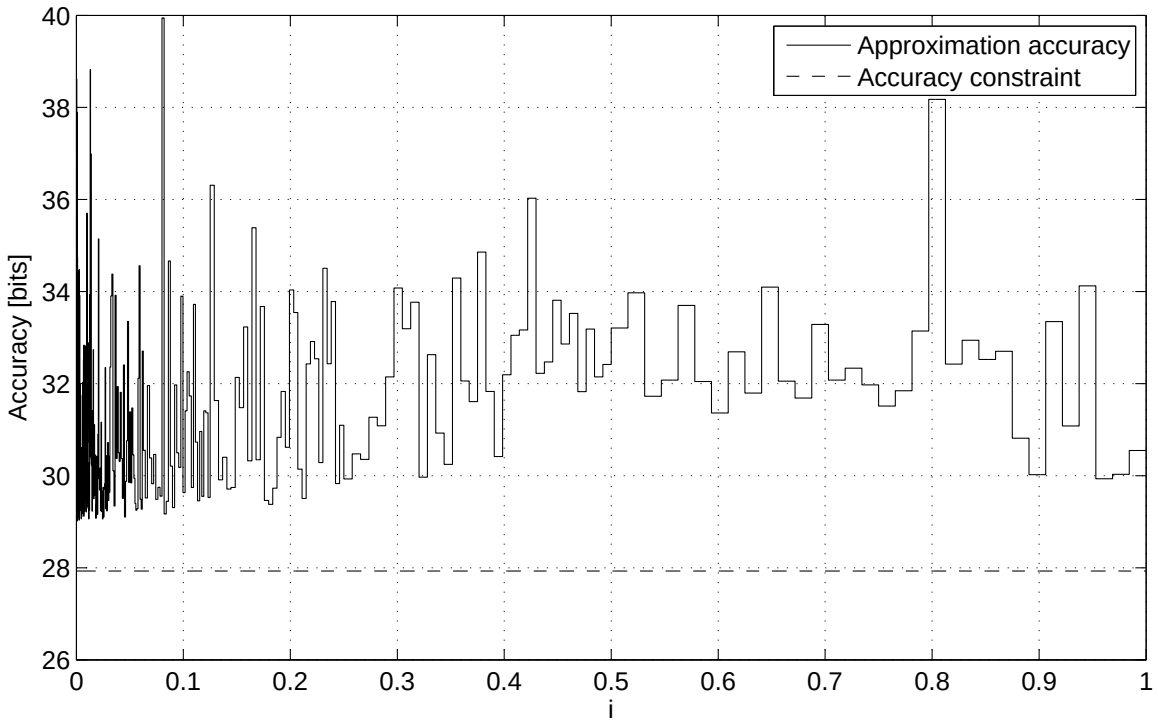

Figure 4. Approximation accuracy (in number of correct bits) for $i \in [2^{-18}, 1[$.

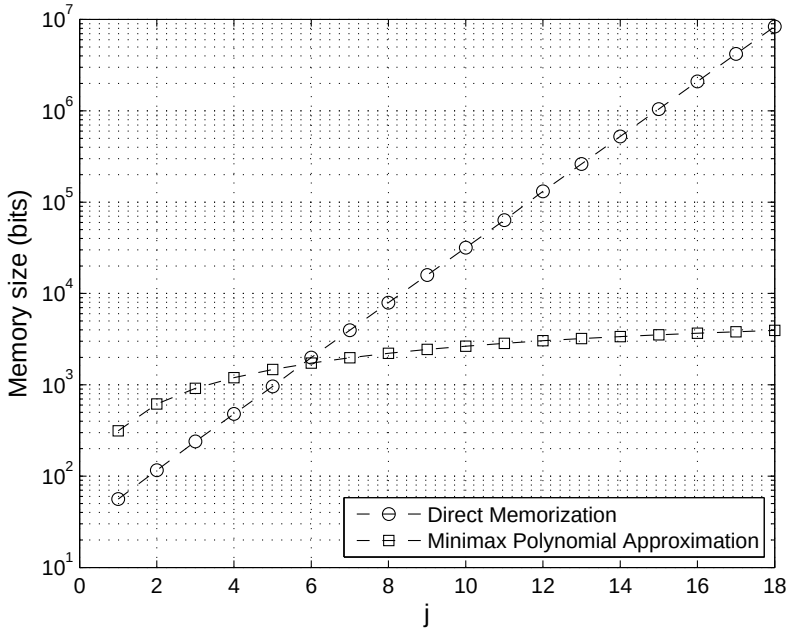

Figure 5. Memory size as a function of the method used to compute the function $H'''(i)$ for $x \in [2^{-18}, 2^{-18+j}[$.

Consequently, the function is approximated for $i \in [2^{-13}, 1[$ and memorized for $i \in [2^{-18}, 2^{-13}[$. As a result, for $i \in [2^{-18}; 2^{-13}[$, the minimal accuracy on the output of (10) is equal 28 correct bits.

After having selected the interval over which the function is segmented and the number of bits required to represent the fractional part of the polynomial coefficients, the size of the data path is investigated. Here, the Horner scheme is used:

$$p^{\star}(x)_{quantized} = ((a_3 \times x + a_2) \times x + a_1) \times x + a_0 \tag{11}$$

For each inner segment, the Horner scheme accuracy is investigated by using Matlab fixed-point simulations. Figure 6 presents the signal flow graph (SFG) used to approximate the function (10) for $i \in [2^{-13}, 1[$ using (11). The data is represented in fixed-point format. Fixed-point data is made-up of an integer part and a fractional part. Fixed-point data format is specified as (b, m, n), where b, m and n are respectively the number of bits (word length) for the data, the integer part and the fractional part. The T block is a truncation of the fractional part. After each multiplication, the number of bits required by the fractional part is investigated. An exhaustive search is performed to find the minimal number of bits required by the fractional part on each multiplication output. The datapath fractional part must be equal to 30.

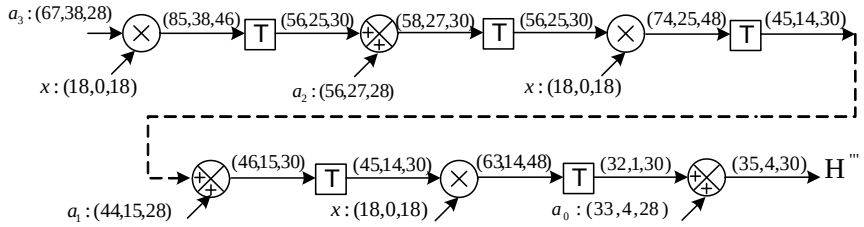

Figure 6. Signal flow graph to compute the function $H^{'''}(i)$ for $x \in [2^{-13}, 2^{1}[$.

The approximation accuracy of $H^{'''}(i)$ for $i \in [2^{-13}, 2^{1}[$ by using the SFG is presented in Figure 7. The minimal precision is equal 27.94 correct bits, which is greater than the targeted accuracy constraint.

4.4. Hardware architecture

One point which has not been discussed is the indexing of the polynomial coefficients. The input i of $H^{'''}(i)$ is a 18-bit data. The 13 most significant bits of i $(i_{17}, \ldots, i_5)$ are used to select the outer segment. A leading zero detector (LZD) detects the position of the leading zero. The solution proposed by[21] and presented in Figure 8 has been selected. A simple OR gate cascade and a 1-bit counter are needed to compute the leading zero position of the 13-bit data. The cascade is applied to the 13 MSBs of i to address the 13 different outer segments. The LZD produces the variable P2SR address which lies in $\{1, \ldots, 13\}$. After having selected the outer segment, the valid inner segment is indexed using the 5 MSBs following the (P2SR address $+ 4)^{\text{th}}$ bit of i. The inner segment is selected using the 5-bit sequence $(i_{15}, i_{14}, ..., i_{11})$ which provides the inner segment position among the 32 different inner segments.

To save the coefficients a_3, a_2, a_1 and a_0, 4 ROMs are used. The ROM depth is equal 416 to memorize all inner segment coefficients. Figure 9 presents the ROM organization for coefficients a_0. The coefficient to select is easily indexed by computing:

$$\text{memory address} = 2^5 \times (\text{P2SR address} - 1) + \text{inner position}. \tag{12}$$

For $i \in [2^{-18}, 2^{-13}[$, the 31 results of $H^{'''}$ are memorized by using a 31×31-bit content-addressable memory.

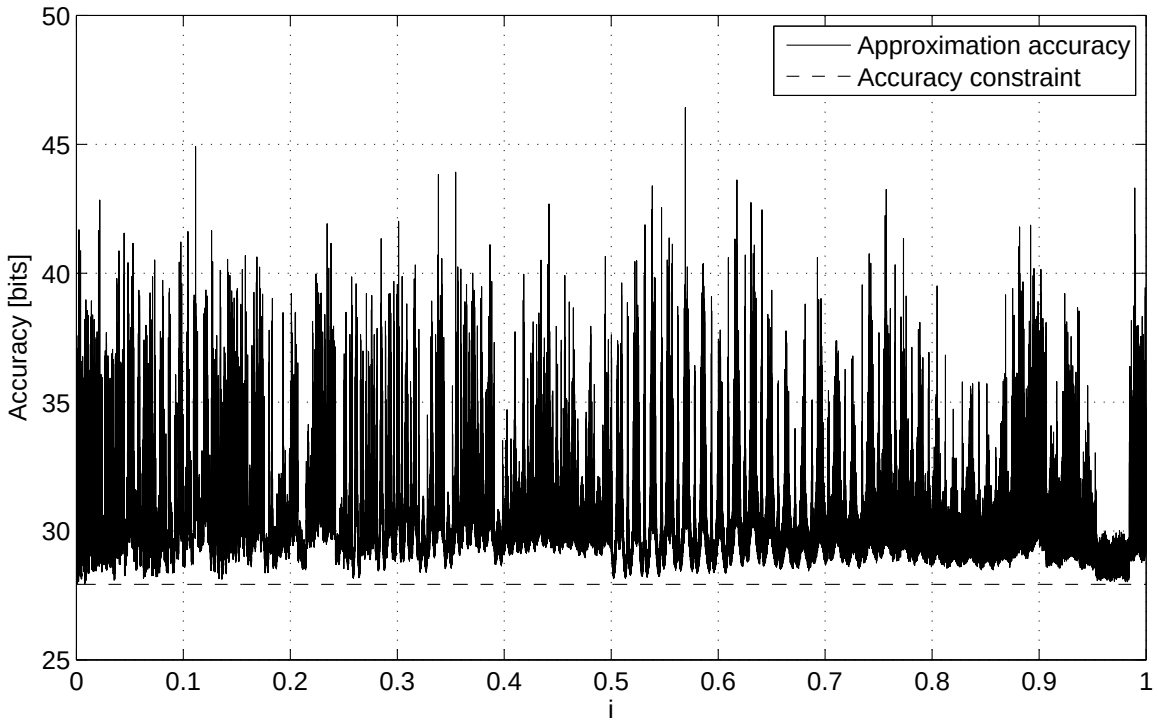

Figure 7. Final approximation accuracy (in correct bits) for $i \in [2^{-13}, 1[$.

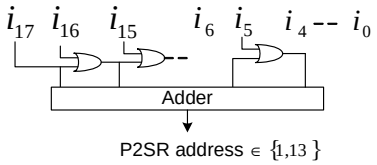

Figure 8. Leading Zero Detector.

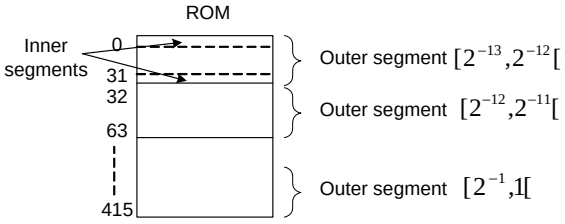

Figure 9. ROM organization.

The design has been synthesized using Simplify 9.0 software for a Xilinx Virtex 5-LX30 FPGA. The maximal frequency of the circuit is equal to 228.4 MHz. The total LUT number is equal 1634 and represents 8% of the FPGA area. The coefficient memory is decomposed respectively into 139 and 194 ROMs of size 128×1 and 256×1. In the worst case, the latency computation is equal to 8 cycles (i.e. 36 ns computation time for a complete approximation).

5. CONCLUSION & FUTURE PROSPECTS

In this paper, we proposed a hardware arithmetic unit dedicated to a statistical test used to evaluate the randomness quality of TRNGs. This test, the entropy one, requires the evaluation of the harmonic series H_i at some rank i. The approximation to the harmonic series usually leads to very large operators. However, our optimized operator uses multiple interval polynomial approximations. The whole domain is decomposed into several small sub-intervals. On each sub-interval, only a low degree polynomial is necessary to perform the approximation with a good accuracy. The proposed method leads to a good trade-off between the degree of the polynomial (cost of multipliers) and the size of the tables required to store the coefficients for all sub-intervals. On a Virtex 5 LX30 FPGA, the circuit runs at 228 MHz, up to 8 cycles are required for the complete approximation and the total area is about 8% of the FPGA.

We plan to automate the proposed method into a VHDL operator generator. This task is not so simple due the possible behaviors of the some target functions such as vertical asymptotes. Then coupling our method to function analysis tools should be a good starting point for this automation. We also plan to target these units for ASICs implementations.

REFERENCES

1. G. Marsaglia, "Diehard: A battery of tests of randomness," tech. rep., Florida State University, 1996.
2. A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and D. Banks, "A statistical test suite for random and pseudorandom number generators for statistical applications," Tech. Rep. 800-22, NIST, 2001.
3. R. Santoro, O. Sentieys, and S. Roy, "On-the fly evaluation of fpga-based true random number generator," in *IEEE Computer Society Annual Symposium on VLSI*, May 2009.
4. R. Santoro, O. Sentieys, and S. Roy, "On-line monitoring of random number generators for embedded security," in *IEEE International Symposium on Circuits and Systems*, May 2009.
5. J.-S. Coron, "On the security of random sources," *LNCS* **1560**, pp. 29–42, 1999.
6. W. Killmann and W. Schindler, "A proposal for: Functionality classes and evaluation methodology for true (physical) random number generators," tech. rep., T-Systems debis Systemhaus Information Security Services and Bundesamt für Sicherheit in der Informationstechnik (BSI), 2001.
7. W. Killmann and W. Schindler, "AIS-31. functionality classes and evaluation methodology for physical random number generators," tech. rep., BSI, 2001.
8. I. Vasyltsov, E. Hambardzumyan, Y.-S. Kim, and B. Karpinskyy, "Fast digital trng based on metastable ring oscillator," in *CHES*, *Lecture Notes in Computer Science* **5154**, pp. 164–180, Springer, 2008.
9. NIST, "Security requirements for cryptographic modules," Tech. Rep. FIPS 140-2, Department of Commerce, May 2001.
10. W. Schindler and W. Killmann, "Evaluation criteria for true (physical) random number generators used in cryptographic applications," in *4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pp. 431–449, Springer-Verlag, 2003.
11. U. M. Maurer, "A universal statistical test for random bit generators," in *CRYPTO '90: 10th Annual International Cryptology Conference on Advances in Cryptology*, pp. 409–420, Springer-Verlag, 1991.
12. J.-S. Coron and D. Naccache, "An accurate evaluation of Maurer's universal test," in *SAC '98: Proceedings of the Selected Areas in Cryptography*, pp. 57–71, Springer-Verlag, (London, UK), 1998.
13. M. D. Ercegovac and T. Lang, *Digital Arithmetic*, Morgan Kaufmann, 2003.
14. J.-M. Muller, *Elementary Functions: Algorithms and Implementation*, Birkhäuser, 2nd ed., 2006.
15. A. Tisserand, "High-performance hardware operators for polynomial evaluation," *Int. J. High Performance Systems Architecture* **1**, pp. 14–23, Mar. 2007.
16. N. Brisebarre, J.-M. Muller, A. Tisserand, and S. Torres, "Hardware operators for function evaluation using sparse-coefficient polynomials," *IEE Electronics Letters* **42**, pp. 1441–1442, Dec. 2006.
17. N. Brisebarre, J.-M. Muller, and A. Tisserand, "Computing machine-efficient polynomial approximations," *ACM Transactions on Mathematical Software* **32**, pp. 236–256, June 2006.

18. E. Remes, "Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation," *C.R. Acad. Sci. Paris* **198**, pp. 2063–2065, 1934.
19. M. B. Villarino, "Ramanujan's harmonic number expansion into negative powers of a triangular number," *J. Inequal. Pure and Appl. Math.* **9**(3), 2008. article 89.
20. D. DeTemple and S.-H. Wang, "Half integer approximations for the partial sums of the harmonic series," *Journal of mathematical analysis and applications* **160**, pp. 149–156, 1991.
21. D.-U. Lee, W. Luk, J. Villasenor, and P. K. Cheung, "Hierarchical segmentation schemes for function evaluation," in *IEEE Conf. on Field-Programmable Technology*, 2003.