Efficient Implementation of Pseudochaotic Piecewise Linear Maps with High Digitization Accuracies

Tommaso Addabbo, Ada Fort, Santina Rocchi, Valerio Vignoli

Department of Information Engineering
University of Siena, 53100 Italy
(e-mail: addabbo@dii.unisi.it)

Davide De Caro, Nicola Petra

Biomedical, Electronic and Telecommunication Engineering Dept.
University of Naples Federico II, 80125 Italy

⋆ RESEARCH MANUSCRIPT ⋆

– PLEASE REFER TO THE PUBLISHED PAPER[1] –

Abstract

In this paper we discuss the efficient implementation of pseudochaotic piecewise linear maps with high digitization accuracies, taking the Rényi chaotic map as a reference. The proposed digital architectures are based on a novel algorithmic approach, that uses carry save adders for the nonlinear arithmetic modular calculations arising when computing piecewise linear maps with a finite precision. As a result, the system can be implemented by digital circuits obtaining high throughputs, which are not dependent on the digital resolution, whereas involving a hardware complexity linearly proportional to the number of bits used for representing the discretized state. The proposed solutions result to be particularly suitable for the implementation of pseudo-random number generators based on pseudo-chaos, or for the definition of efficient digital blocks that can be integrated in most of the pseudo-chaotic cyphers proposed in the literature.

1 Introduction

In the last decades, the irregular and unpredictable behavior of chaotic dynamical systems has stimulated a large amount of researches focusing on topics pertaining Information Technology, e.g., for random number generation, for the definition of cryptographic protocols, for the design of stochastic sources

[1]International Journal of Circuit Theory and Applications, 2010, vol. 40, p. 1-14.

with predefined statistical characteristics to be used in telecommunication systems [1–4].

It is well known that chaos can not take place in deterministic dynamical systems having discrete and finite phase spaces, since several properties that characterize the chaotic dynamics (e.g., the aperiodicity) can not be exhibited. Nevertheless, since the sixties the study of chaos has been kept up with simulations on computers, and it was only many years later that chaos simulation has been analyzed from an algorithmic information theory point of view [5–7]. In the last few years, several authors have proposed the use of *discretized* (or *digitized*) chaotic systems for different applications, ranging from pseudo-chaotic cryptography to the generation of pseudo-random numbers (see, e.g., [1,8,19,20] and references cited therein). Kocarev et al. in [9,10] proposed a formal definition of *Discrete Chaos*, providing a theoretical link between the world of discrete-valued dynamical systems and some of the features defined for continuous phase space chaotic systems, such as the Lyapunov exponents (see also [11]). The theory discussed in these papers was stimulated by a well known evidence: for reasonably high digital resolutions in the computations, computer simulations can produce reliable descriptions of the ergodic behavior of chaotic systems [11, 12]. As a result, several aspects of the dynamical behavior of a chaotic system can be approximated by the discrete-valued dynamics of a finite state machine (FSM), with an accuracy that depends on the digital resolution of the machine itself (see, e.g., [13] and references cited therein). Adopting this point of view, several pseudo-chaotic cryptographic cyphers or pseudo-random number generators were proposed in literature exploiting the emulation of the 'confusing' and 'diffusing' dynamics of certain chaotic systems. Moreover, as far as pseudo-random number generation is concerned, digitized nonlinear systems may not suffer from the regularity problems that affect the sequences generated exploiting linear numerical methods [8].

Until now much research has been focused on the theoretical study of pseudochaotic systems, whereas a limited effort was devoted to investigate the digital implementation of such nonlinear circuits. Indeed, this issue has been faced case by case by reusing traditional digital arithmetic circuits for the direct design of the functional blocks defining the digitized nonlinear maps taken into account. As a typical result, as also shown in this paper, for the obtained solutions the implementation complexity (in terms of required chip area and time-delay) becomes unsustainable even for limited digital resolutions (e.g., 64 bits), with an area consumption that increases according to a polynomial law of the digitizing accuracy.

In this paper we discuss the efficient implementation of pseudochaotic piecewise linear maps with high digitization accuracies, and referring to not-trivial values of the slopes of the branches[2]. Taking as a reference the digitized Rényi chaotic map proposed in [8], we propose digital architectures based on a novel algorithmic approach that uses carry save adders for the nonlinear arithmetic modular calculations arising when computing piecewise linear maps with a finite precision. As a result, the FSM can be implemented by digital circuits obtaining high throughputs, which are not dependent on the digital resolution, whereas

[2]When integer values of the slopes of the branches are taken into account, digitized piecewise linear maps defined over dyadic rationals are often equivalent to systems combining two or more linear congruential generators, whose hardware implementation is based on traditional modular arithmetic circuits [16, 21].

involving a hardware complexity linearly proportional to the number of bits used for representing the discretized state. The proposed hardware architectures result to be particularly suitable for the implementation of pseudo-random number generators based on pseudo-chaos (as proposed in [8]), or for the definition of efficient digital blocks that can be integrated in most of the pseudo-chaotic cryptographic cyphers proposed in literature (see, e.g., [1, 14]), referring to digital resolutions in the order of those used today in cryptography (i.e., from 128 to 2048 bits). Despite the general applicability of the proposed approach, in this paper we focus on the Rényi chaotic map for two reasons: first, in [8] it was highlighted the important peculiarity of this special system, according to which by varying the system parameters a wide family of nonlinear permutations is defined, that can be used for the definition of cryptographic ciphers or reliable pseudo-random number generators. Second, the analytical expression obtained when digitizing the Rényi map is the simplest, whereas it can be considered as a starting point for the implementation of any piecewise linear map.

The paper is organized as in the following. In Section 2 we introduce the notation and some preliminary remarks about the digital implementation of the digitized Rényi map, discussing the problems that may arise when adopting a standard direct design approach. Moreover, in this section we define the complexity model adopted for evaluating and comparing the different solutions that are proposed in the paper. In Sections 3, 4 and 5 we discuss in detail how to efficiently implement the discretized Rényi map. For this purpose, different assumptions about the map parameters, that can be considered fixed or user-selectable, were taken into account, and an original algorithmic approach for the computation of the digitized map using carry save adders is discussed. Conclusions and references close the paper.

2 Hardware Implementation of the Discretized Rényi Chaotic Map

In this section we introduce the notation and some preliminary remarks about the digital implementation of the pseudo-chaotic dynamical system derived from the Rényi chaotic map proposed in [8].

Let us consider the Rényi chaotic map $T_\beta : [0, 1) \to [0, 1)$ defined as

$$T_\beta(x) = \beta x \bmod 1, \tag{1}$$

where $1 < \beta \in \mathbb{R}$, $\lfloor \cdot \rfloor$ is the floor function and, given two non-negative real numbers a, b, it results $a \bmod b = a - \lfloor a/b \rfloor\, b$. We remark that the parameter β in (1) can assume any real value greater than one, i.e., any number of branches of the chaotic map is taken into account.

Referring to the set of integers $\Lambda_n = \{k \in \mathbb{N}; 0 \leq k < 2^n; n > 0\}$, in [8] a digitization strategy for the map T_β was proposed, obtaining the nonlinear function $f : \Lambda_n \to \Lambda_n$

$$f(k) = q2^{n-i}k + \left\lfloor k2^{-i} \right\rfloor \bmod 2^n, \tag{2}$$

where n is the number of bits used for representing the digitized state, $0 < i < n$, and q is an odd positive integer such that $q2^{n-i} < 2^n$. By setting $b = q2^{n-i} \in$

n	Area ($\mu m^2 \times 10^3$)	Max. Clock Freq. (MHz)	Max. Throughput (Gbit/s)
32	66	286	9.152
64	217	182	11.648
128	832	154	19.712
256	3765	133	34.048

Table 1: Results for the direct implementation of (3) synthesized in the TSMC $0.18\mu m$ CMOS technology.

Λ_n, the expression (2) can be written as

$$f(k) = \left(bk \bmod 2^n + \left\lfloor k2^{-i} \right\rfloor \bmod 2^n\right) \bmod 2^n. \tag{3}$$

As proved in [8], by varying n, q, i in (2) we obtain a family of nonlinear permutations $\{\phi : \Lambda_n \to \Lambda_n\}$ that can be defined as *pseudo-chaotic permutations* [9, 10].

The traditional direct implementation of (3) can lead to very fast solutions[3], involving an area consumption that increases according to a square law of the numbero of digitization bits n. This is typical for architectures implementing the direct modular multiplication $b \cdot k \bmod 2^n$, as reported in Table 1, in which the map $f(\cdot)$ was synthesized in the TSMC $0.18\mu m$ CMOS technology (throughput = computed next state bit per second).

It is clear from this example that even for 128 resolution bits the direct implementation approach results to be unsustainable from an area consumption point of view, obtaining excessively high throughputs that have scarce practical meaning in most of the applications. For these reasons in this paper we adopt a different design strategy, studying architectures which trade-off the throughput and the consumed circuit area, taking into account very high values of n.

Since for any digital resolution n the hardware implementation of (3) depends on which quantities among k, b, i are assumed to be variable (user selectable) inputs, we distinguish between the case in which the inputs are k, b, i (Section 3) and the case in which i is constant and a priori known, whereas k and b are user-changeable inputs (Section 4). Even if in the latter case the cardinality of the parameter space is reduced[4], this constrain can greatly improve the performance of the circuit, as it is discussed in Sec. 4. From a design point of view, refering to the theoretical results presented in [8], the choice between the two cases depends on the target application, taking into account the differences between the proposed architectures in terms of involved complexity and achieved throughput

[3] I.e., the design of a customized digital circuit for computing $k_{n+1} = f(k_n)$ in one clock-cycle.

[4] In detail, when i is fixed to any value $0 < i < n$ the cardinality of the parameter space is 2^{i-1}, whereas if i is variable the cardinality is $2^n - 1$ [8].

2.1 Notation

In what follows, we refer to the binary representation of integers, i.e., if $a \in \Lambda_n$, the n-tuple of bits $(a_{n-1}, \ldots, a_0)$ is such that $a = \sum_{j=0}^{n-1} a_j 2^j$. Using digital hardware the function $(\cdot) \bmod 2^n$ can be implemented by simply taking the n least significant bits of the binary representation of the argument, and setting all the remaining most significant bits equal to 0. Accordingly, if A is a generic register storing the integer A we write $B|_n = A$ for denoting this latter operation, i.e., the integer stored in B is equal to $A \bmod 2^n$ and its size is assumed to be exactly n-bit. On the other hand, if $F(A, C)$ is a logical function based on the two registers A and C, when we write $B|_n = F(A, C)$ the computation of $F(A, C)$ is restricted to those minimal calculations required for obtaining the n least significant bits of the result. If A is a n-bit register, we write $A \ll l$ for denoting the standard l-bit left shift of A, that is the hardware calculation of $A2^l \bmod 2^n$. Similarly, we write $A \gg l$ for denoting a standard l-bit right shift of A, which corresponds to the hardware calculation of $\lfloor A2^{-l} \rfloor$. We denote with A_j the j-th bit stored in the register A, being A_0 the least significant bit and, for avoiding confusion in the notation, we distinguish the usual arithmetical operations from the logical operators `or`, `and` and `xor`, denoted with the symbols $\oplus$, $\odot$ and $\otimes$, respectively.

2.2 Complexity model

To evaluate and compare different methods for calculating (3), we refer to the well known methodology adopted in [15, 16], i.e., the implementation effort is expressed in terms of the product AT (chip Area, Time delay). According to this approach, we assume the unit of measurement for the time delay as the delay of one full adder, whereas from the chip area consumption point of view a 1-bit register and a full adder are quantified as 4 area units and 8 area units, respectively. Accordingly, the theoretical results discussed in this paper have to be correctly interpreted when applied for the design of real circuits, as discussed in the following.

As it is understood in [16], the rough estimation of the AT product makes sense when different design approaches are compared letting the size of the processed variables go to infinity. This is our case, if we let the digital resolution involved in the discretization of the Rényi map tend to infinity. Furthermore, we want to stress that the methodology we adopt for estimating the AT product must be interpreted as a sketchy tool to perform a first order comparison of different digital design approaches. Indeed, the actual amount of resources necessary for implementing a given digital circuit strongly depends on the adopted technology, on the degrees of freedom in the customization of the digital layout design, or on the size of the available logical blocks (e.g., in FPGAs). For this reason, all the results presented in this paper have been validated performing the syntheses of the digital circuits, referring to a $0.18\mu m$ CMOS technology.

Algorithm A1

1. $P = 0$; $Q = k$;
2. **for** $j = (n-1) \rightarrow 1$
3. $\quad P|_n = P \ll 1 + b_j k$;
4. $\quad$ **if** $j \geq n - i$ **then**
5. $\quad\quad Q|_n = Q \gg 1$;
6. $P|_n = P \ll 1 + Q$;

Figure 1: A basic algorithm for computing the function (3), obtaining $P = f(k)$.

3 Digital implementation of (3) with i variable

Assuming the state k stored in a n-bit register, the product $b \cdot k$ is equal to

$$\sum_{j=0}^{n-1} k_j 2^j \cdot \sum_{u=0}^{n-1} b_u 2^u = \sum_{j=0}^{n-1} \sum_{u=0}^{n-1} k_j b_u 2^{j+u}. \tag{4}$$

By noting that the representation of $b = q2^{n-i}$ has its least significant $n-i$ bits set to 0 (i.e., the bits $b_{n-i-1}, \ldots, b_0$), the previous product is given by

$$b \cdot k = \sum_{j=0}^{n-1} \sum_{u=n-i}^{n-1} k_j b_u 2^{j+u}. \tag{5}$$

Accordingly, by assuming P, Q and k as n-bit registers, a basic algorithm for computing the quantity in eq. (3) $P = f(k)$ is reported in Fig. 1. This basic algorithm is a modified version of the popular interleaved modular multiplication algorithm for computing xy mod M, in which the $(\cdot)$ mod M operation is performed automatically by simply taking in each addition the n least significant bits of the result [16].

The loop (A1.3)-(A1.5) is repeated $n-1$ times since the least significant bit of $b = q2^{n-i}$ is 0 regardless of any allowed i value, with $0 < i < n$. For the execution flow of this algorithm two counting circuits are required: one for counting the n additions and one for counting i right-shift of the register Q, for obtaining the term $2^{-i}k$.

REMARK: In the following, for analyzing and discussing the implementation effort required by any method for computing (3), we neglect those circuits involved in the algorithm flow control logic (among which the counting circuits). As discussed in Sec. 5, this simplification does not invalidate the proposed analysis and the consequent results, as it was also confirmed by the synthesis of the digital circuits.

The critical operation in this algorithm is the n-bit addition required in steps (A1.3) and (A1.6), performed n times. The left shift of P in (A1.6) is necessary since the loop (A1.3)-(A1.5) is repeated only $n-1$ times, instead of n times. The standard additions can be performed, e.g., with a n-bit ripple carry adder (requiring an AT complexity equal to $8n^2$), or with a carry look ahead adder (whose complexity grows as $n \log^2 n$) [16]. Assuming to use a ripple carry adder for the additions in (A1.3) and (A1.6), the AT complexity required for the calculation of (3) with Algorithm A1 can be roughly estimated as $24n^3$. The

total count is obtained considering: the area for the n-bit registers P and Q ($8n$); the area for the two n-bit registers k and b ($8n$) used for storing the state k and the factor $b = q2^{n-i}$; the area of the ripple carry adder ($8n$); the time delay required for completing the operations (n^2, which is given by n additions requiring a time-delay of n each). As it can be noted by inspecting the result, the standard addition operation has a critical weight in the overall AT product.

It is worth pointing out that the algorithm A1 computes the function (3) using a sequential multiplier. The same computation could be obtained using a parallel approach [17], which has some advantages in terms of speed and power dissipation when the algorithm is converted into a circuit. However, since, as shown in the previous section, we are interested in high-precision discretized maps, the value of the parameter n is assumed to be very high ($10^2 - 10^3$), and parallel architectures would result in a huge, impractical amount of hardware resources. For this reason, in the following, we only refer to *sequential architectures*.

3.1 An improved computation of (3) with i variable

A popular choice for improving the time delay performance of a multiplier (which is obtained by repeated additions) is the use of carry save (CS) adders [16]. An n-bit CS adder is a parallel ensemble of n full-adders without any horizontal connection. Its main function is to add three integers X, Y and Z to produce two integers C and S such that $C + S = X + Y + Z$.

Assuming X, Y, Z to be three n-bit registers, the bits of S and C are calculated using the equations

$$S_j = X_j \otimes Y_j \otimes Z_j, \tag{6}$$

$$C_{j+1} = (X_j \odot Y_j) \oplus (X_j \odot Z_j) \oplus (Y_j \odot Z_j), \tag{7}$$

for $j = 0, \ldots, n-1$, whereas $C_0 = 0$. Accordingly, the AT complexity of a CS adder is $8n$, and we use the notation $C, S = X + Y + Z$ for denoting the operation of calculating C and S, with $S < 2^n$ and $C < 2^{n+1}$.

Since

$$(C + S) \bmod 2^n = (C \bmod 2^n + S \bmod 2^n) \bmod 2^n, \tag{8}$$

we can obtain the result of the above summation by simply considering the n least significant bits of C and S, respectively. According to the introduced notation we denote this latter operation with $C, S|_n = X + Y + Z$. It is worth noting that since $C_0 = 0$, all the information about the quantity $C \bmod 2^n$ is stored in $n - 1$ bits.

Using CS adders, the Algorithm A1 is modified as reported in Fig. 2, in which C, S, P and k are assumed to be n-bit registers, whereas R is a one-bit register. The final result is obtained using a serial adder to add C and S, whereas the (A1.2)-(A1.5) instructions in Algorithm A1 are transformed in the operations (A2.2)-(A2.5). Summarizing the overall AT complexity required for the calculation of (3) with Algorithm A2 can be roughly quantified as $56n^2+16n$. The total count is obtained considering the area for the registers C, S, P, k and b ($20n$), the area of the CS adder ($8n$), the area of the serial adder (8), and the time delay required for the calculation $2n$.

Algorithm A2

1. $C = 0; S = 0; P = k; R = 0;$
2. **for** $j = (n-1) \rightarrow 1$
3. $\quad C, S|_n = C \ll 1 + S \ll 1 + b_j k;$
4. $\quad$ **if** $j \geq n - i$ **then**
5. $\quad\quad P|_n = P \gg 1;$
6. $C, S|_n = C \ll 1 + S \ll 1 + P;$
7. **for** $j = 0 \rightarrow (n-1)$
8. $\quad P_j = C_j \otimes S_j \otimes R;$
9. $\quad R = (C_j \odot S_j) \oplus (C_j \odot R) \oplus (R \odot S_j);$

Figure 2: The algorithm for computing function (3) using CS adders, obtaining $P = f(k)$.

We want to stress that the involved time-delay is *linearly proportional* to the number of bits used for representing the digitized state of the chaotic systems. As a consequence, the ratio between the number of computed digitized state bits and the time-delay of the computation does not depend on n. Hence, as confirmed in the next subsection, the throughput is not influenced by n.

3.2 Synthesis of a FSM based on the Algorithm A2

According to the schematic block diagram depicted in Fig. 3, the authors designed a digital FSM computing the trajectory $\{k_0, f(k_0), \ldots, f^m(k_0), \ldots\}$ triggered by the initial condition k_0. The obtained sequence, divided by 2^n, emulates the evolution of the Rényi chaotic map

$$T_\beta(x) = (q2^{n-i} + \frac{1}{2^i})x \bmod 1, \tag{9}$$

assuming $x_0 = k_0 2^{-n}$ as the initial condition. At the end of each computation cycle, the FSM stores the state $k_{m+1} = f^{m+1}(k_0)$ in the same register k that contained the previous state $k_m = f^m(k_0)$. The digital design was synthesized using a $0.18\mu m$ CMOS technology, and the proposed solution allowed to reach a 0.312Gbit/s throughput, independently from the size n of the state registers.

In the designed FSM, given $k_m = f^m(k_0)$, the calculation of the state $k_{m+1} = f^{m+1}(k_0)$ is completed in n clock cycles, assuming for simplicity n even. At the beginning of each computation cycle, the machine initializes to 0 the n-bit registers C, S and the one-bit register R, whereas the n-bit registers k and P contain the previously computed state, or they are initialized to the external seed k_0 if the FSM is starting-up. In this block diagram, k, P, C and S are n-bit registers with enabled twice right-shift, parallel load, and reset. The registers k and P can also twice right-shift their contents while writing their most significant 2 bits. A counting system[5] (not reported in the block diagram) is initialized (to count n and i) and the b factor is loaded and stored in a register B (not reported) that is cyclically left-shifted by two positions, so that after $n/2$ clock cycles its content is back to its initial configuration. The execution flow of the

[5]The aspects related to the counting circuits are discussed in Sec. 5.

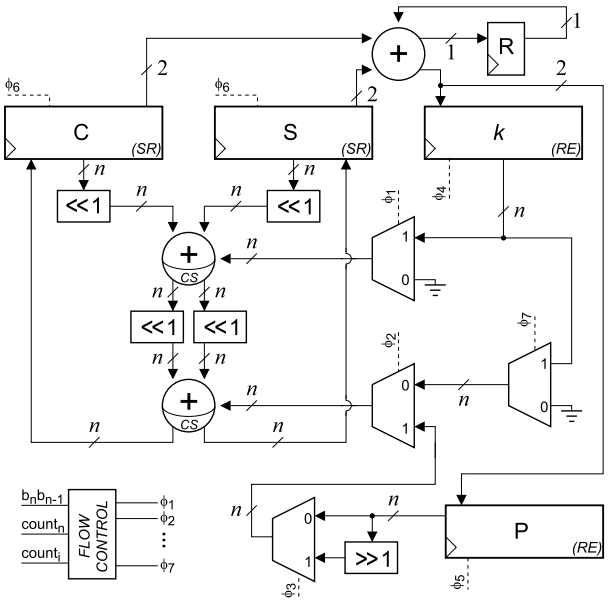


Figure 3: Schematic core-architecture of a FSM exploiting Algorithm A2.

algorithm is regulated by a trivial control logic that drives the four multiplexers and determines the behavior of the special registers C, S, k and P.

In our design in the first $n/2$ clock cycles the machine performs the instructions (A2.2)-(A2.6), performing two CS additions for each clock cycle and storing the result in C and S. At the same time, in each clock cycle the register P is right-shifted by two positions until its content is equal to k shifted by $2 \cdot \lfloor i/2 \rfloor$ positions. On the $n/2$-th clock cycle the second CS addition is executed using for the term $k \gg i$ the content of P, which is further right-shifted in case i is odd (instruction (A2.6)).

In the second $n/2$ clock cycles the machine serially performs the addition (A2.7)-(A2.9), adding two bits in each clock-cycle and progressively writing the result both in k and P. The standard 2-bit ripple-carry adder takes as inputs the two least significant bits of C and S, and the carry bit stored in R. The result is written in each clock cycle in the two most significant bits of k, being in the meanwhile the contents of both this register and the registers C, S twice right-shifted. With this architecture, the time delay related to the critical path can be reasonably supposed not dependent on n.

In order to validate our analysis we have synthesized the circuit of Fig. 3 for n equal to $\{128, 256, 512, 1024, 2048\}$. The synthesis results are summarized in Table 2. According to the synthesis processes, with this architecture the time delay related to the critical path in the timing analysis was the time delay of the cascade of one flip-flop, one multiplexer and two full-adders. As a result the clock-period resulted to be not dependent on n, as expected. The area of the circuit increases almost linearly with n, whereas the maximum clock frequency remains unchanged. The data in the second column (Area) reported in Table 2 do not account for the low-complexity algorithm flow control logic circuits (among which the counting circuits, discussed in Section 5).

n	Area ($\mu m^2 \times 10^3$)	Max. Clock Freq. (MHz)	Max. Throughput* (Gbit/s)
128	71	625	0.312
256	141	625	0.312
512	294	625	0.312
1024	601	625	0.312
2048	1150	625	0.312

Table 2: Synthesis results for circuit of Fig. 3, implemented in the TSMC $0.18\mu m$ CMOS technology.

Algorithm B1

1. $C = 0; S = 0; R = 0;$
2. **for** $j = (n-1) \rightarrow (n-i)$
3. $\quad C, S|_n = C \ll 1 + S \ll 1 + b_j k;$
4. $C, S|_n = C \ll (n-i) + S \ll (n-i) + k \gg i;$
5. **for** $j = 0 \rightarrow (n-1)$
6. $\quad P_j = C_j \otimes S_j \otimes R;$
7. $\quad R = (C_j \odot S_j) \oplus (C_j \odot R) \oplus (R \odot S_j);$

Figure 4: The algorithm for computing $P = f(k)$, based on (3), assuming i constant.

4 Digital implementation of (3) with i constant

If the parameter i is constant and a priori known, then the algorithm A2 can be simplified in the algorithm B1, described in Fig 4. The AT complexity associated with this algorithm can be roughly estimated as $(28n+8)(n+i+1) = 28n^2 + 4n(7i+9) + 8(i+1)$. The total count is obtained considering the area for the registers C, S, P, k, b $(20n)$, the area of the CS adder plus the area of the serial adder $(8n+8)$, and the time delay required for the calculation $n+i+1$. It is worth noting that the term $k \gg i$ in (B1.4) can be obtained with a fixed wired standard right-shift of k by i positions.

4.1 An enhanced alternative approach

We present an alternative approach for the computation of (3), reducing the time delay involved by the Algorithm B1. In the Algorithm B1, the instructions (B1.5)-(B1.7) require a standard serial n-bit adder that involves a time delay of n. Alternatively, we propose an enhanced solution in which the calculation (B1.5)-(B1.7) is performed using CS adders only, assuming to store and represent the state of the digitized system (i.e., the n-bit integer k) *with redundancy*.

In detail, we propose to use two n-bit registers C and S such that

$$k = (C + S) \bmod 2^n. \tag{10}$$

As made clearer in the following, the improvement is obtained performing the calculations using only the registers C and S (that is using only CS adders).

Let us first notice that, given C and S such that (10) holds, the computation of the term $k \gg i$ in (B1.4) on the basis of the knowledge of C, S and i is not trivial. Indeed we have

$$\begin{aligned}\left\lfloor \frac{k}{2^i} \right\rfloor &= \left\lfloor \frac{1}{2^i} \left(\sum_{j=0}^{n-1} (C_j + S_j) 2^j \bmod 2^n \right) \right\rfloor = \\ &= \left\lfloor \sum_{j=0}^{n-1} (C_j + S_j) 2^{j-i} \bmod 2^{n-i} \right\rfloor = \\ &= \left(\sum_{j=i}^{n-1} (C_j + S_j) 2^{j-i} + \lfloor R \rfloor \right) \bmod 2^{n-i},\end{aligned} \tag{11}$$

where $\lfloor R \rfloor \in \{0, 1\}$ since

$$0 \leq R = \sum_{j=0}^{i-1} (C_j + S_j) 2^{j-i} \leq 2 - 1/2^{i-1}. \tag{12}$$

It follows that the computation of the quantity $k \gg i$ on the basis of the knowledge of C and S requires that the *complete addition* of the terms C and S has to be performed, since no short cut exists that allows for neglecting the i least significant bits of C and S.[6] The only exception is given when $C = 0$ (or, alternatively, $S = 0$). In such a special case, assuming $C = 0$, we have $k = (C + S) \bmod 2^n = S$ and $k \gg i = S \gg i$. The complete addition of the terms C and S requires a slow standard adder, and that is what we have to avoid to improve the performance of the implementation.

4.2 Special repeated mod 2^n-additions using CS adders

In this subsection we present a simple theoretical result on the basis of which the computation of $k \gg i$ can be obtained in a FSM using CS adders only. To this aim, let us consider the sequence $\{(A(m), B(m)), m \in \mathbb{N}\}$ where

$$A(m), B(m)|_n = A(m-1) + B(m-1) + 0; \tag{13}$$

and such that $(A(0)+B(0)) \bmod 2^n = \theta \in \mathbb{N}$, with $0 \leq \theta < 2^n$. By construction we have that $(A(m)+B(m)) \bmod 2^n = \theta$ for all $m \in \mathbb{N}$. We have the following

Proposition 1 *For any $A(0), B(0) \in \{0,1\}^n$ if $m \geq n$ then $B(m) = (A(0) + B(0)) \bmod 2^n$ and $A(m) = 0$.*

[6] Indeed, the most significant bits of the result depend on the least significant ones.

Proof. According to (6), (7) and (13) for any $m \geq 0$ we have

$$B_j(m+1) = B_j(m) \otimes A_j(m), \tag{14}$$
$$A_{j+1}(m+1) = B_j(m) \odot A_j(m), \tag{15}$$

with $A_0(m+1) = 0$. Thus, we first note that if $A(m) = 0$ then $A(m+q) = 0$ and $B(m+q) = B(m)$ for all $q > 0$. Let us now assume $0 < A(m) < 2^n$ and let p be the greatest integer such that $0 \leq p \leq n-1$ and

$$A(m) = \sum_{j=p}^{n-1} A_j(m) 2^j, \tag{16}$$

with $A_p(m) = 1$ and, in general, $A_j(m) \in \{0, 1\}$. Using (15) we have $A_{j+1}(m+1) = 0$ for $0 \leq j \leq p-1$, that is

$$A(m+1) = \sum_{j=p+1}^{n-1} A_j(m+1) 2^j. \tag{17}$$

Accordingly, if $p = n-1$ then $A(m+1) = 0$; on the other hand, regardless of $A(m)$, in the worst case no more than n time-steps are necessary for assuring $A(m+n) = 0$ and, consequently, $A(m+j) = 0$ for all $j \geq n$. The proof is completed noting that by construction we have $(A(m) + B(m)) \bmod 2^n = (A(0) + B(0)) \bmod 2^n = \theta$ for all $m \in \mathbb{N}$, and therefore if $A(m) = 0$ then $B(m) = (A(0) + B(0)) \bmod 2^n$. □

In other words, the previous proposition shows that if n repeated $\bmod 2^n$-additions (13) are performed using CS adders, then for all $m \geq n$ the term $B(m)$ contains exactly the summation $(A(0) + B(0)) \bmod 2^n$. Moreover, it is interesting noting that if $A_0(0) = 0$ then we have that $A(m) = 0$ for all $m \geq n-1$, i.e., one time-step before than the general case considered in the proposition. This happens when $A(0)$ and $B(0)$ are the result of a previous addition performed using a CS adder. Concluding this subsection, we stress that the operations (14)-(15) can be obtained using a parallel ensemble of n *half adders*, that involve a halved delay with respect to a CS adder.

4.3 An enhanced algorithm fully based on CS adders.

On the basis of the result provided by Proposition 1, in this subsection we present the algorithm of Fig. 5, fully based on CS adders.

In this algorithm, C, D, S, T are assumed to be n-bit registers, being C and S initialized to two values C_0 and S_0 such that $k = (C + S) \bmod 2^n$ (e.g., we can chose $C_0 = 0$ and $S_0 = k$). As made clearer in the following, by properly choosing the two parameters w and h the loop of instructions (B2.2)-(B2.5$_h$) equivalently executes the partial multiplication in (B1.2)-(B1.3). It is worth noting that, according to the previous theoretical result, inside the loop (B2.2)-(B2.5$_h$) the instructions (B2.5$_1$)-(B2.5$_h$) do not change the value of the sum $k = (C + S) \bmod 2^n$, even if the binary contents of the registers C and S are modified. This operation represents the *core of the enhanced algorithm*: at each time-step in the partial result of the multiplication the correct term $k = (C + S) \bmod 2^n$ is added, even if the binary content of the registers C

Algorithm B2

1. $D = 0; T = 0; C = C_0; S = S_0;$
2. **for** $j = (n-1) \rightarrow (n-w)$
3. $\quad D, T|_n = C \ll 1 + S \ll 1 + b_j S;$
4. $\quad D, T|_n = D + T + b_j C;$
5$_1$. $\quad C, S|_n = C + S + 0;$
$\vdots$ $\quad\quad\quad \vdots$
5$_h$. $\quad C, S|_n = C + S + 0;$
6. $D, T|_n = D \ll (n-w) + T \ll (n-w) + S \gg i;$

Figure 5: The enhanced algorithm for computing (3). If C_0 and S_0 satisfy (10), the algorithm computes D and T such that $(D+T) \bmod 2^n = f(k)$. Moreover, after the calculations, $C = 0$ and $S = (S_0 + C_0) \bmod 2^n = k$.

and S is progressively modified (according to the sequence (13) discussed in the previous proposition). In a synchronous clock-edge-triggered digital circuit, the two serial instructions (B2.3)-(B2.4) and the serial instructions (B2.5$_1$)-(B2.5$_h$) can be performed *in parallel*. The time-delay associated to (B2.3)-(B2.4) is roughly equal to that one of two full-adders (i.e., 2), whereas the time-delay associated to (B2.5$_1$)-(B2.5$_h$) is roughly equal to that one of h half-adders, that is $\approx h/2$. Accordingly, if we set $1 \leq h \leq 4$, the time delay of the parallel structure computing simultaneously (B2.3)-(B2.4) and (B2.5$_1$)-(B2.5$_h$) is roughly equal to 2.

As discussed in the following, the two numbers w and h must be chosen such that when exiting the loop all the information about the system state k is stored in S, with $C = 0$. According to the Proposition 1 and the subsequent discussion, the minimum number of special halved CS additions $C, S|_n = C + S + 0$ such that $S = C + S$ and $C = 0$ is $n - 1$. Therefore, the numbers w and h in the Algorithm B2 must be chosen such that

$$\begin{cases} i \leq w \leq n-1, \\ wh \geq n-1, \\ 1 \leq h \leq 4. \end{cases} \tag{18}$$

The above constraints are determined according to the following reasoning. In order to correctly calculate the product $b \cdot k$, at least i iterations are required, since b can have its i most significant bits not equal to zero: this implies $i \leq w \leq n-1$. In order to empty the binary content of the register C, at least $n-1$ halved CS additions are necessary. Since in each loop h special halved CS additions are performed, we must have $wh \geq n-1$. Finally, the last condition $1 \leq h \leq 4$ was introduced to limit the time delay of the parallel structure computing (B2.5$_1$)-(B2.5$_h$) to the time required by the execution of (B2.3)-(B2.4).

Given (18), in order to minimize the overall time-delay of the loop (B2.2)-(B2.5$_h$), that is proportional to w, one has to choose w such that

$$w = \max\left\{i, \left\lceil \frac{n-1}{4} \right\rceil\right\}. \tag{19}$$

Once determined w, the area involved by the special halved CS adders required

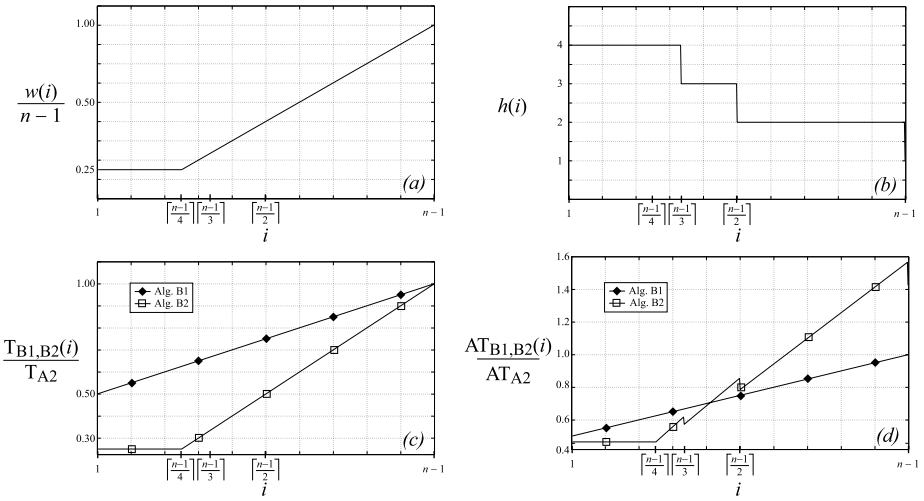


Figure 6: *(a)* – the optimum loop iteration number w as a function of i, normalized to the quantity $n-1$; *(b)* – the optimum number of halved CS adders h as a function of i; *(c)* – the time-delay normalized to the time-delay of the Algorithm A2; *(d)* – the AT complexity normalized to the AT complexity of the Algorithm A2.

in (B2.5$_1$)-(B2.5$_h$) can be minimized by choosing

$$h = \left\lceil \frac{n-1}{w} \right\rceil = \left\lceil \frac{n-1}{\max\left\{i, \left\lceil \frac{n-1}{4} \right\rceil\right\}} \right\rceil . \tag{20}$$

The instruction (B2.6) concludes the algorithm performing the operation (B1.4) and completing the computation of registers D and T, which contain the next-state values for C and S.

The overall AT complexity involved by Algorithm B2 can be quantified as $4n(9+h)(2w+1)$. The total count is obtained considering the area for the registers C, S, D, T and b ($20n$), the area of the two CS adders (16n) plus the area required by the special halved CS adders ($h4n$) and the time delay required for the calculation $2w+1$.

The four subplots in Fig. 6 report, respectively: (a) the optimum loop iteration number w as a function of i, normalized to the quantity $n-1$; (b) the optimum number of halved CS adders h as a function of i; (c) the time-delay normalized to the time-delay of the Algorithm A2; (d) the AT complexity normalized to the AT complexity of the Algorithm A2. As it can be seen by the subplot (c), the time-delay involved by the Algorithm B2 is always lower than or equal to that one estimated for the Algorithm B1. Even the product AT results to be smaller if i is approximately lower than $0.4n$.

4.4 Synthesis of a FSM based on the Algorithm B2

According to the schematic block diagram depicted in Fig. 7, the authors designed a digital FSM based on the algorithm B2 for computing the trajectory $\{k_0, f(k_0), \ldots, f^m(k_0), \ldots\}$ triggered by the initial condition k_0. The digital

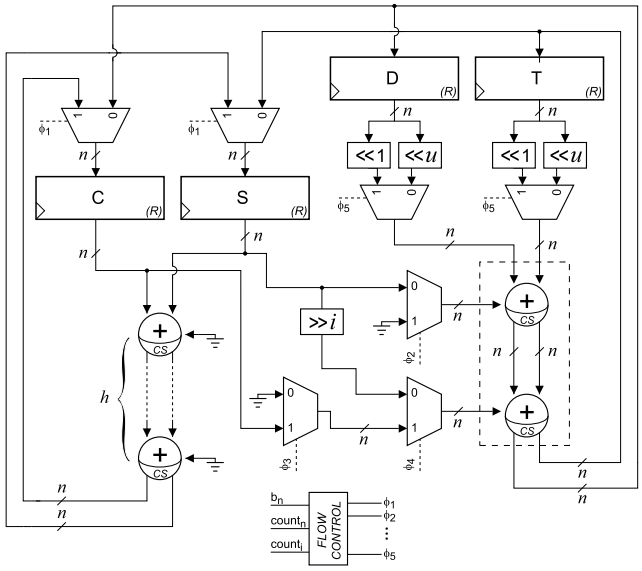


Figure 7: Schematic core-architecture of a FSM exploiting Algorithm B2. In the scheme, $u = n - w$.

design was synthesized using a $0.18\mu m$ CMOS technology, and the proposed solution allowed to easily reach a maximum clock frequency higher than 620MHz almost independently from the size n of the state registers.

Assuming at the start-up $C = 0$ and $S = k_0$, after the m-th computation cycle the FSM stores in the registers C and S two values such that $(C + S) \bmod 2^n = f^m(k_0)$. In the first phase of the computation cycle, according to the instructions (B2.3)-(B2.4), the registers D and T are used for progressively storing the carry-save equivalent representation of the product $b \cdot k$, computed using two arrays of full-adders. At the same time, according to the instructions (B2.5_1)-(B2.5_h), at each clock cycle the registers C and S are added using h arrays of half-adders.

After w clock cycles the registers D and T contains the final *carry* and *sum* components of the term $kb2^{w-n}$, whereas the binary content of the registers C and S is equal to 0 and k, respectively[7]. The final result can therefore be obtained CS-summing together $D \ll (n-w)$, $T \ll (n-w)$ and $S \gg i$, according to the instruction (B2.6). This sum is implemented by re-using the two array of n full-adders in the dashed box of Fig.7. To this aim, three multiplexers are used in Fig.7 to dynamically change the inputs of the full-adders.

The circuit in Fig.7 has been synthesized in the TSMC $0.18\mu m$ CMOS technology, for n equal to $\{128, 256, 512, 1024, 2048\}$ and assuming $i = \left\lceil \frac{n-1}{4} \right\rceil = \frac{n}{4}$. Accordingly, in this specific case, $\frac{n}{4}$ clock cycles are necessary for computing the next-state. The synthesis results reported in Table 3 show that also for this circuit the maximum clock frequency does not depend on n, whereas the area of the circuit increases almost linearly with n. We want to stress that the time delay related to the critical path in the timing analysis is the time delay of the

[7]That is, the correct n-bit representation of the current state $k = (C + S) \bmod 2^n$ can be outputted by the FSM after w clock cycles from the beginning of each computation cycle.

n	Area ($\mu m^2 \times 10^3$)	Max. Clock Freq. (MHz)	Max. Throughput* (Gbit/s)
128	101	625	1.250
256	210	625	1.250
512	436	625	1.250
1024	913	621	1.242
2048	1694	621	1.242

Table 3: Synthesis results for the circuit of Fig.7, implemented in the TSMC $0.18\mu m$ CMOS technology and assuming $i = n/4$.

cascade of one flip-flop, one 2:1 multiplexer and two full-adders, i.e., the time delay of the critical path does not depend on i[8]. Accordingly, the maximum clock frequencies reported in Table 3 are typical for this kind of architecture. On the other hand, the required area is dependent on i, since this parameter determines the number of halved CS adders. Also in this case the data reported in Table 3 do not account for the low-complexity algorithm flow control logic circuits.

4.5 Comparison of synthesis results

The synthesis results reported in Tables 2 and 3 show that the circuit in Fig.7 requires about a 50% larger area than the circuit in Fig.3. On the other hand, the circuit in Fig.7 requires one fourth of the number of clock cycles needed by the circuit in Fig.3 to compute the next-state in the sequence $\{k_0, f(k_0), \ldots, f^m(k_0), \ldots\}$ (see Fig.6.*c*). Hence, the throughput of the circuit in Fig.7 is four times greater than the throughput of the circuit in Fig.3. In conclusion, the overall AT complexity of the circuit in Fig.7 is approximately $1.5 \times 0.25 \approx 0.4$ times the AT complexity of the circuit in Fig.3. This result is in accordance with the data of Fig.6.*d*.

5 The counting circuits

The execution flow of any of the algorithms A1-B2 requires specific digital circuits for counting the algorithmic execution steps, and for recording the progress of the computation task (e.g., in the Algorithm A2 the counting of both n and i is required). Therefore, since the counting process is functional to the correct operation of any digital circuit discussed before, solutions that cause a reduction of the throughput must be avoided (i.e., solutions that lower the maximum allowed clock frequency).

[8]Indeed, the parameter i determines both the number of iterations of the loop (B2.2)-(B2.5_h) and the number of halved CS adders, not affecting the critical path.

There are several architectural solutions proposed in literature for enhancing the speed of counters. An interesting solution was proposed by Ercegovac and Lang in [18], adopting the prescaling technique of long counters. According to this approach, the counter is partitioned in a series of sub-blocks of increasing size, and the counting rate can be made equal to the time-delay of one half adder, that is *not dependent* on the counter size [18]. Accordingly, adopting proper strategies for the design of the counting circuits, the theoretical results presented in the previous sections have a general validity. This assumption has been validated by synthesizing the circuits of Fig.3 and Fig.7 with and without the counting system, obtaining the same critical path in both cases, as expected[9].

6 Conclusions

In this paper we have discussed the efficient implementation of pseudochaotic piecewise linear maps with high digitization accuracies (e.g., using for the representation of the digitized state a number of bits in the magnitude order of $10^2 - 10^3$), taking the Rényi chaotic map as a reference. We have proposed different architectural solutions, and we have discussed a novel algorithmic approach for the computation of the digitized map using carry save adders. As a fundamental result of this work, the proposed solutions allow for obtaining high throughputs which are not dependent on the digital resolution of the finite state machine. For the calculation of the digitized map (3), we have distinguished between the case in which the inputs are k, q, i (Section 3) and the case in which i is known and assumed constant, whereas k and q are user-changeable inputs (Section 4). In this latter case, on the basis of the theoretical result present in subsection 4.2, we have proposed an enhanced architecture fully based on CS adders which can considerably lower the time-delay involved by the calculations, obtaining – for specific values of i – a reduced overall AT complexity.

Starting from the design of the digitized Rényi map, the proposed algorithmic approach can be generalized for implementing any digitized piecewise linear map in which fixed-point finite-precision modular calculations are involved.

References

[1] J. Amigó, L. Kocarev, and J. Szczepanski, "Theory and practice of chaotic cryptography," *Physics Letters A*, vol. 366, pp. 211–216, 2007.

[2] S. Ergün and S. Özoğuz, "Truly random number generators based on non-autonomous continuous-time chaos," *International Journal of Circuit Theory and Applications*, vol. 38, no.1, pp. 1–24, Feb. 2010.

[3] P. Li, W. Zhang, Z. Li, W. Liu and W. A. Halang, "FPGA implementation of a coupled-map-lattice-based cryptosystem," *International Journal of Circuit Theory and Applications*, vol. 38, no.1, pp. 85–98, Feb. 2010.

[9] It is worth noting that for the highest digitizing resolution considered in this paper (i.e., 2048 bits) a low-complexity properly prescaled 11 bit binary counter suffices.

[4] R. Rovatti, G. Mazzini and G. Setti, "Memory-m Antipodal Processes: Spectral Analysis and Synthesis," *IEEE Transaction on Circuits and Systems – I*, vol. 56, no.1, pp. 156–167, Feb. 2009.

[5] T. Krüger and S. Troubetzkoy, "Complexity, randomness, discretization: some remarks on a program of J.Ford," *Physica D*, vol. 105, pp. 97–104, 1997.

[6] T. Krüger, P. Seibt, and S. Troubetzkoy, "Complexity and randomness of recursive discretization of dynamical systems," *Random and Computational Dynamics*, vol. 2, pp. 1–21, 1994.

[7] B. Chiricov and V. F., "An algorithmic view of pseudochaos," *Physica D*, vol. 129, pp. 223–235, 1999.

[8] T. Addabbo, M. Alioto, A. Fort, A. Pasini, S. Rocchi, and V. Vignoli, "A class of maximum-period nonlinear congruential generators derived from the Rényi chaotic map," *IEEE Transactions on Circuits and Systems - I*, vol. 54, no. 4, pp. 816–828, 2007.

[9] L. Kocarev, J. Szczepanski, J. Amigó, and I. Tomovski, "Discrete chaos–I: Theory," *IEEE Transaction on Circuits and Systems – I*, vol. 53, no. 6, pp. 1300–1309, June 2006.

[10] J. Amigó, L. Kocarev, and I. Tomovski, "Discrete entropy," *Physica D*, vol. 228, pp. 77–85, 2007.

[11] T. Krüger and S. Troubetzkoy, "Ergodic theory on recursive lattices: a theoretical approach to computer simulations of chaotic systems." in *Proceedings: Dynamics of Complex and Irregular Systems*. World Scientific, Ph. Blanchard ed., 1993.

[12] B. Coomes, H. Kocak, and K. Palmer, "Shadowing in discrete dynamical systems," in *Six Letures on Synamical Systems*, B. Aulbach and F. Colonius, Eds. World Scientific, 1996, pp. 163–208.

[13] S. Li, G. Chen, and M. Xuanqin, "On the dynamical degradation of digital piecewise linear chaotic maps," *International Journal of Bifurcation and Chaos*, vol. 15, no. 10, pp. 3119–3151, 2005.

[14] T. Xiang, X. Liao, G. Tang, Y. Chen, and K. Wong, "A novel cryptosystem based on iterating a chaotic map," *Physics Letters A*, vol. 349, pp. 109–115, 2006.

[15] M. Shimmler and V. Bunimov, "Area-time optimal modular multiplication," *Mathematics of Computation*, vol. 44, pp. 519–521, 1985.

[16] C. Koc, "Rsa hardware implementation," RSA Laboratories, RSA Data Security, Inc., Tech. Rep., 1995.

[17] V. G. Oklobdzija, D. Villeger, and S. S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," *IEEE Transactions on Computers*, vol. 45, no. 3, pp. 294–306, Mar. 1996.

[18] M. Ercegovac and T. Lang, "Binary counter with counting period of one half adder independent of counter size," *IEEE Transaction on Circuits and Systems*, vol. 36, no. 6, pp. 924–926, June 1989.

[19] T. Lin and L.O. Chua, "A new class of pseudo-random number generator based on chaos in digital filters," *International Journal of Circuit Theory and Applications*, vol. 21, no.5, pp. 473–480, Oct. 1993.

[20] M. Alioto, S. Bernardi, A. Fort, S. Rocchi and V. Vignoli, "An efficient implementation of PRNGs based on the digital sawtooth map," *International Journal of Circuit Theory and Applications*, vol. 32, no.6, pp. 615–627, Oct. 2004.

[21] P. L'Ecuyer, "Uniform Random Number Generation," *Annals of Op Research*, vol. 3, no.1, pp. 77–120, 1994.