Bulletin of the National Research Centre

RESEARCH Open Access

FPGA design and implementation for adaptive digital chaotic key generator

Ghada Elsayed[1*], Elsayed Soleit[1] and Somaya Kayed[2]

Abstract

Background Information security is very important in today's digital world, especially cybersecurity. The most common requirement in securing data in all services: confidentiality, digital signature, authentication, and data integrity is generating random keys. These random keys should be tested for randomness. Hardware security is more recommended than software. Hardware security has more speed and less exposure to many attacks than software security. Software security is vulnerable to attacks like buffer overflow attacks, side-channel attacks, and Meltdown–Spectre attacks.

Results In this paper, we propose an FPGA Implementation for the adaptive digital chaotic generator. This algorithm is proposed and tested before. We introduce its implementation as hardware. This algorithm needs a random number seed as input. We propose two designs. The first one has an input random number. The second one has PRNG inside. The target FPGA is Xilinx Spartan 6 xc6slx9-2-cpg196. We used MATLAB HDL Coder for the design. We propose a configurable Key block's length. For 32 bit the maximum frequency is 15.711 MHz versus 11.635 MHz for the first and second designs respectively. The area utilization of the Number of Slice Registers is 1% versus 2%. The number of Slice Look Up Tables is 40% versus 59%. number of bonded input output blocks is 64% versus 66%. otherwise are the same for the two designs.

Conclusions In this paper, we propose an efficient and configurable FPGA Design for adaptive digital chaotic key generator. Our design has another advantage of storing the output keys internally and reading them later.

Keywords FPGA, MATLAB HDL Coder, Chaotic

Background

Nowadays, generating a cryptographic random number generator is very important. Its importance comes from information security importance, cyber security, and testing machine learning. How to generate pseudo-random numbers is a question to which we'll pick an answer of it and introduce a hardware Implementation to it over the FPGA. The FPGA Design is performed by MATLAB HDL Coder (MathWorks 2023). Its workflow is shown in Fig. 1. In Fig. 1 there are two input methods either by Simulink or Matlab script. Our work is based on a Matlab script. MATLAB HDL Coder needs two files; the first one is the top level of the design and the other is its test. After creating the HDL CODER project, we select only, the top function of the design and its tester. Then the HDL CODER converts this script to an HDL file. The HDL file is then forwarded to the FPGA's vendor synthesizer. The generation of the HDL code from the MATLAB script requires understanding the I/O types. This is necessary for the FPGA pins. It also requires making all types compatible with each other. It also requires identifying all the used functions and libraries such that all functions could be mapped into hardware. Functions like printf and scanf are examples of not supported functions. The MATLAB HDL Coder as a

*Correspondence:
Ghada Elsayed
Ghada.farouk@eng.mti.edu.eg
[1] Electrical Engineering Department, Modern University for Technology and Information, Cairo, Egypt
[2] Electrical Department, Obour High Institute for Engineering and Technology, Obour, Egypt

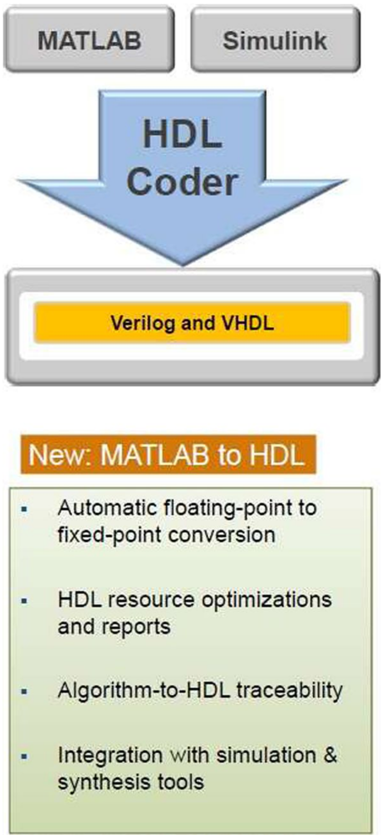

MATLAB
Simulink
HDL Coder
Verilog and VHDL
New: MATLAB to HDL
Automatic floating-point to fixed-point conversion
HDL resource optimizations and reports
Algorithm-to-HDL traceability
Integration with simulation & synthesis tools

Fig. 1 MATLAB HDL Coder Workflow (Eda techchannel OpenSystems Media 2023)

tool for FPGA Design was a subject for evaluation by many researchers. They first recommended using it for only fast proofing of the idea. This was the result of area utilization and speed compared to the VHDL is not encouraging (Elsayed and Kayed 2022). However, the latest research has proposed an Optimization technique to dramatically, improve the result (Kayed and Elsayed 2022). Based on that recommendation we here design and implement the key generation over the FPGA using MATLAB HDL Coder. We selected the new chaotic random numbers key generator (Soleit 2018) which was published by Elsayed A. Soleit T 2018. Soleit is one of the authors of this paper so, we already have a tested Algorithm that can generate a cryptographic random number generator written and tested as a MATLAB script. Figure 2 shows the Block Diagram of the new adaptive chaotic key generator (Soleit 2018). this was already, written by MATLAB script in previous research, and fortunately, the author of this research is one of the authors of this paper, Fig. 3 illustrates the given simulation code. Therefore, the code is one of the givens of this research. We can notice that the input depends on the rand (x) function which is built in MATLAB. All the scripts can

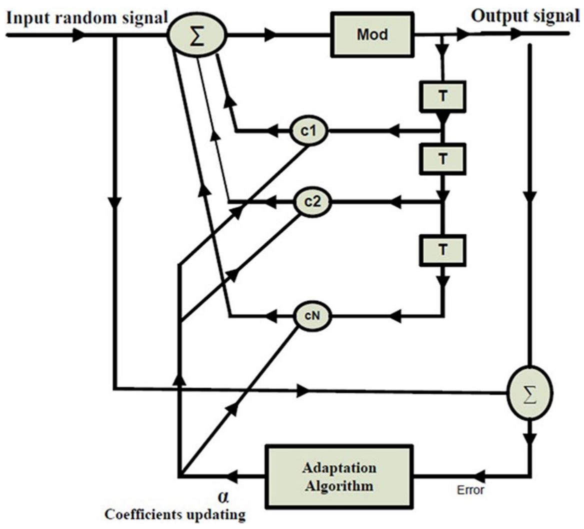


Fig. 2 New adaptive chaotic key generator block diagram

be hardware implemented except this function. In Fig. 2 we can see the adaptive coefficients from c1 to cN, here we selected N to be three to just prove the idea. However, the design is configurable and we can easily change Nc, where Nc is the number of the coefficients − 1. This block diagram is the representation of the following equations:

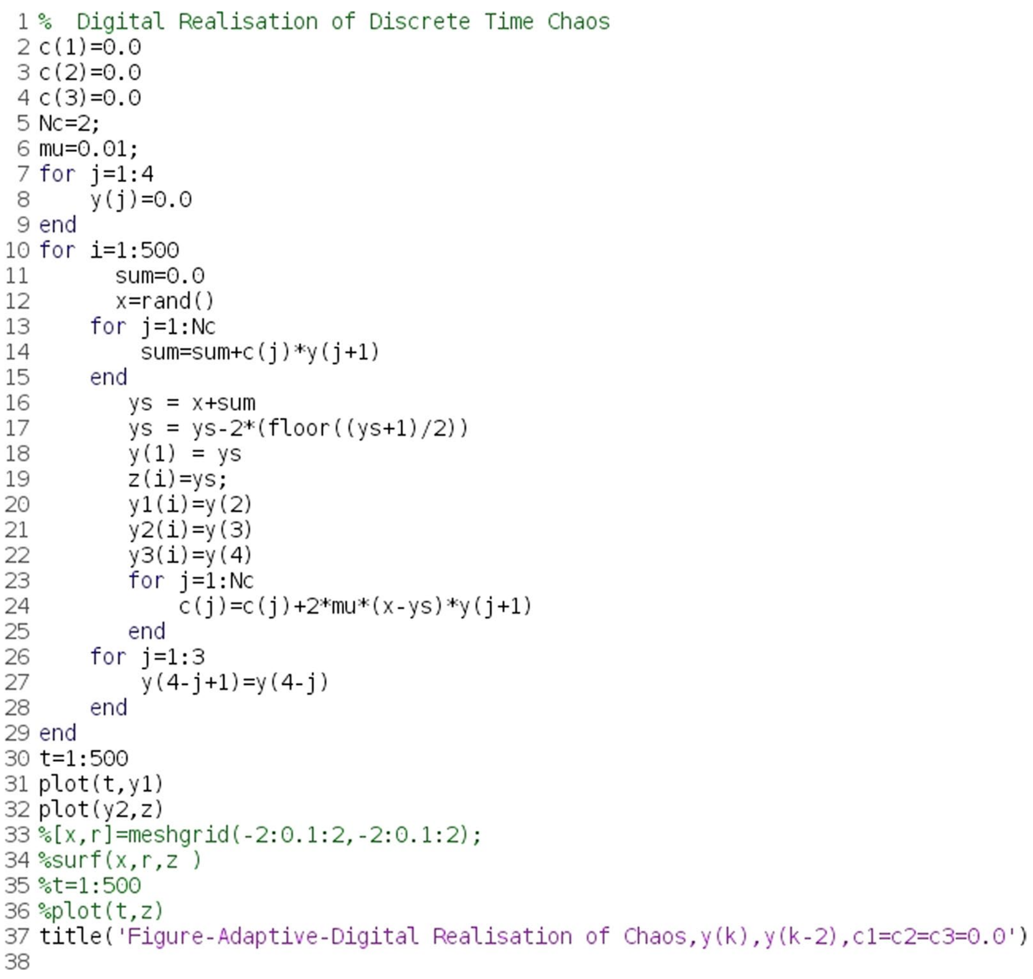
$$y_k = Z_k + \sum_{i=0}^{N} C_i y_{k-i} \tag{1}$$

$$\mathrm{mod}\ (y) = y - 2 \times \lfloor \frac{y+1}{2} \rfloor \tag{2}$$

$$y_k = \mathrm{mod}\left(Z_k + \sum_{i=0}^{N} C_i y_{k-i} \right),\ y_i \in I = (-1, 1) \tag{3}$$

where Zk is the seed, in the simulation code illustrated in Fig. 3, it is stored in variable x and initialized by rand (). Again, the Rand function can't be implemented over the FPGA. Rand itself should be replaced with a method that generates its output. To generate a random seed we adopted a published algorithm in (Soleit 2018). The Hardware Implementation of this part is subject to another publication. This algorithm depends on the accuracy of calculating the square root and gives some sort of randomness in the fractional part. The first order is represented by Eq. (4) and the second order is represented by Eq. (5). This algorithm was also already both simulated and tested.

$$x_{i+1} = \sqrt{(x_i)}\ \mathrm{mod}\ 1 \times 10^{n} \tag{4}$$



```
% Digital Realisation of Discrete Time Chaos
c(1)=0.0
c(2)=0.0
c(3)=0.0
Nc=2;
mu=0.01;
for j=1:4
    y(j)=0.0
end
for i=1:500
      sum=0.0
      x=rand()
    for j=1:Nc
        sum=sum+c(j)*y(j+1)
    end
       ys = x+sum
       ys = ys-2*(floor((ys+1)/2))
       y(1) = ys
       z(i)=ys;
       y1(i)=y(2)
       y2(i)=y(3)
       y3(i)=y(4)
       for j=1:Nc
           c(j)=c(j)+2*mu*(x-ys)*y(j+1)
       end
    for j=1:3
        y(4-j+1)=y(4-j)
    end
end
t=1:500
plot(t,y1)
plot(y2,z)
%[x,r]=meshgrid(-2:0.1:2,-2:0.1:2);
%surf(x,r,z )
%t=1:500
%plot(t,z)
title('Figure-Adaptive-Digital Realisation of Chaos,y(k),y(k-2),c1=c2=c3=0.0')
```

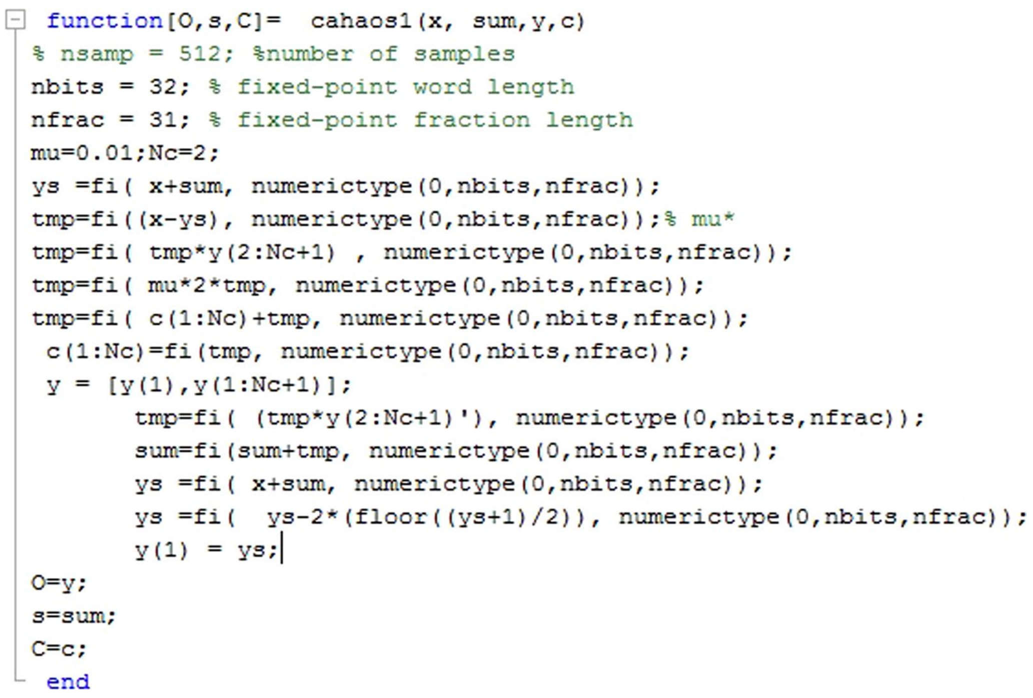
Fig. 3 Original MATLAB Code

$$x_{i+1} = \sqrt{(x_i + x_{i-1})} \bmod 1 \times 10^n \tag{5}$$

We here used the second order (Su and McSweeney 2019) to supply our main algorithm with the required random input. In the following section, we will illustrate the design process from the starting point of the simulated algorithm by Matlab script to the implementation of the FPGA.

Methods

Chaotic Random Number generator Design is done using MATLAB HDL Coder. The Block Diagram of Fig. 2 was written by MATLAB just for simulation and testing. As we mentioned its code is introduced in Fig. 3, some modifications should be done to the code the make it suitable to be valid for the generation of the HDL code. Figure 4 illustrates the modified code. We used the fixed-point numeric object to unify almost all the types. Values of nbits and n fractions are word length and fraction length respectively. The memory size is configurable by the variable nsamp. We here put the Nc variable equal to 2 in the code and this implies three coefficients c1, c2, and c3. This is the main function that takes the input random number and the feedback 's' for sum and 'C' for coefficients. The outputs are the random number, also used for feedback, and the two feedback feed the inputs with sum and c. This function itself needs to be embedded into another function to connect the feedback with the inputs, this function is called Cahaos. Figure 5 shows the top view of the Cahaos design. The Figure illustrates the inputs and the outputs. We have a 32-bit input random number as a seed and we have output to the generated chaotic random number. The output is also 32-bit. We have also a clock and clock enable input pins, CLK and clk_enable respectively. We have also input reset and output enable ce_out. As recommended by Kayed and Elsayed (2022) we used the persistent keyword to generate registers on the feedback signal. This improves both the area and speed. In Fig. 6 the code of this function is illustrated. After defining the persistence for the



```
function[O,s,C]=  cahaos1(x, sum,y,c)
% nsamp = 512; %number of samples
nbits = 32; % fixed-point word length
nfrac = 31; % fixed-point fraction length
mu=0.01;Nc=2;
ys =fi( x+sum, numerictype(0,nbits,nfrac));
tmp=fi((x-ys), numerictype(0,nbits,nfrac));% mu*
tmp=fi( tmp*y(2:Nc+1) , numerictype(0,nbits,nfrac));
tmp=fi( mu*2*tmp, numerictype(0,nbits,nfrac));
tmp=fi( c(1:Nc)+tmp, numerictype(0,nbits,nfrac));
 c(1:Nc)=fi(tmp, numerictype(0,nbits,nfrac));
 y = [y(1),y(1:Nc+1)];
        tmp=fi( (tmp*y(2:Nc+1)'), numerictype(0,nbits,nfrac));
        sum=fi(sum+tmp, numerictype(0,nbits,nfrac));
        ys =fi( x+sum, numerictype(0,nbits,nfrac));
        ys =fi(  ys-2*(floor((ys+1)/2)), numerictype(0,nbits,nfrac));
        y(1) = ys;
O=y;
s=sum;
C=c;
 end
```

Fig. 4 Cahaos1 function's code after modification

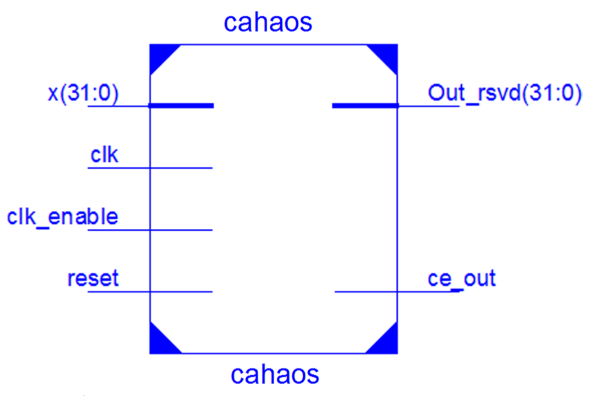
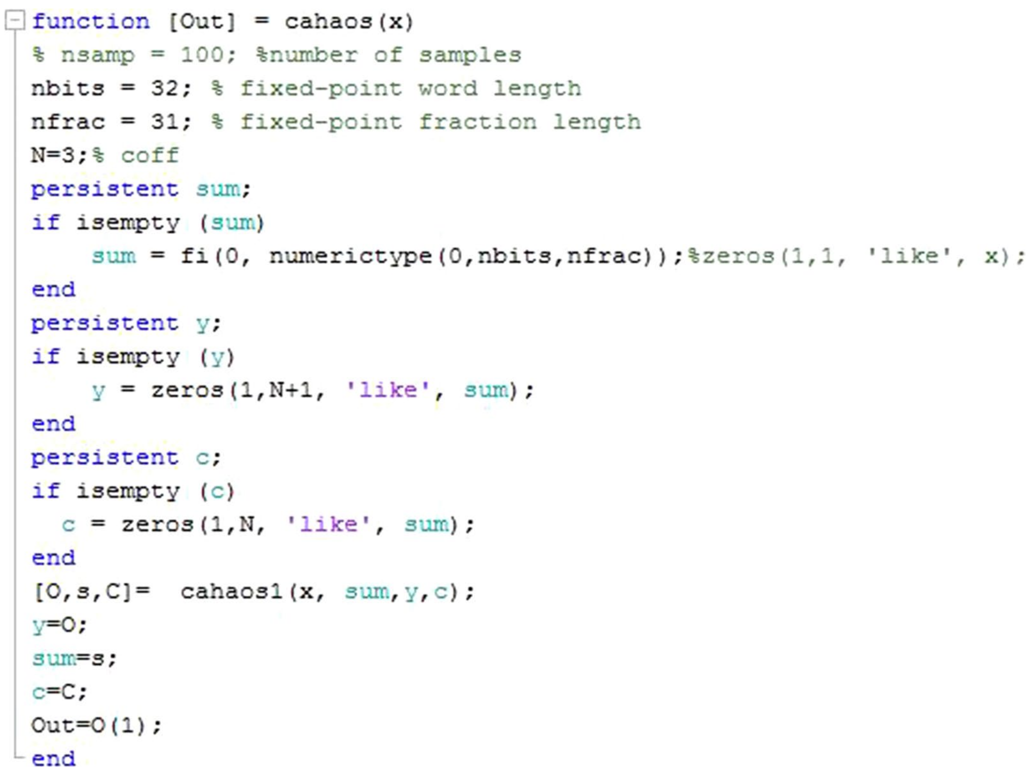


Fig. 5 Cahaos Top view

three feedback, we just call the cahaos1 function and then connect the inputs with the outputs The second file needed by the MATLAB workflow is the tester. Figure 7 illustrates the code for the tester. Here we call the cahaos function 512 times. As we can see all variables are configurable and we can easily change them. The tester output is illustrated in Fig. 8. For that, the design is implemented. Frequency and its area utilization are discussed in section five. Now we will introduce the design by adding a module that generates a random seed. The new design is called cahaos_sqrt_rand_top. So instead of feeding the design input with a random number we just feed it with a non-square number. This non-square number is go through equ 5 to generate the required random seed. The design of this part is out of the scope of this paper. We also supported the design with a memory to store the generated output and we can read this memory by activating the input read signal Rd. In Fig. 9 the top view of cahaos_sqrt_rand_top is illustrated. We here output the valid output that indicates the finishing of the square part. Also, the LOAD/CALC input is used for loading the input non-square number and then starting to calculate the square root. These were the inputs and the outputs different from the previous design. The code and its testing are illustrated in Figs. 10 and 11 respectively.

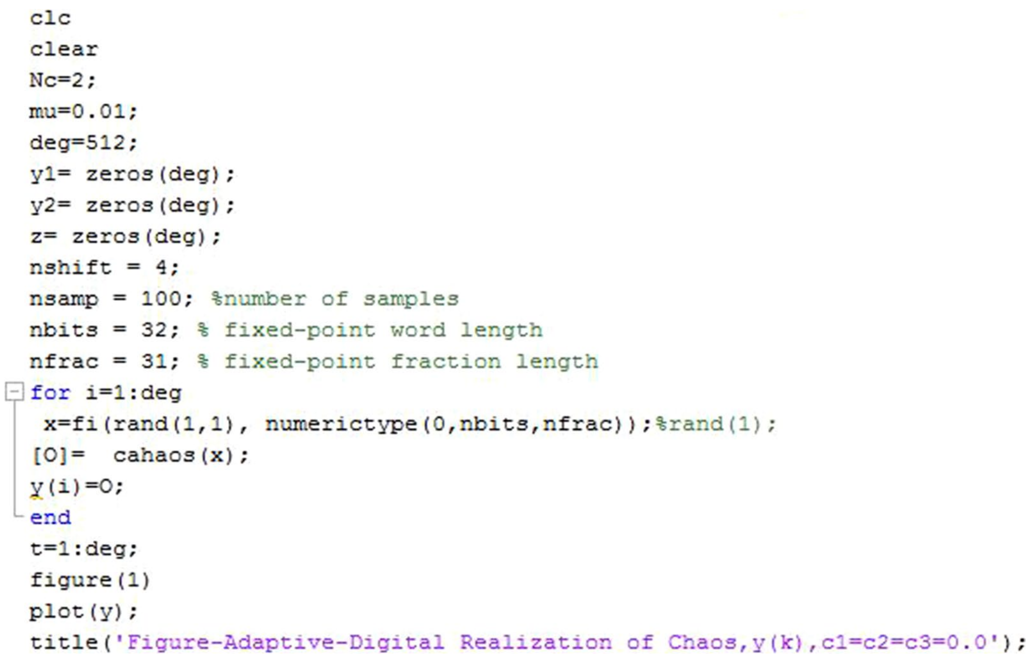
Results

The Target Device is Xilinx Spartan 3 xc6slx9-2-cpg196 FPGA (Digilent 2023). The estimated values for device utilization for design 1 and design 2 are illustrated in Tables 1 and 2 respectively. For 32 bit the maximum frequency is 15.711 MHz versus 11.635 MHz for the first and second designs respectively. The area utilization of the Number of Slice Registers is 1% versus 2%. The number of Slice LUTs is 40% versus 59%. The number of fully used LUT-FF pairs is 5% versus 8%. The number of bonded IOBs is 64% versus 66%. The number of BUFG/BUFGCTRL/BUFHCEs and the Number of DSP48A1s are the same for the two designs 6% and 100%. In this



```
function [Out] = cahaos(x)
% nsamp = 100; %number of samples
nbits = 32; % fixed-point word length
nfrac = 31; % fixed-point fraction length
N=3;% coff
persistent sum;
if isempty (sum)
    sum = fi(0, numerictype(0,nbits,nfrac));%zeros(1,1, 'like', x);
end
persistent y;
if isempty (y)
    y = zeros(1,N+1, 'like', sum);
end
persistent c;
if isempty (c)
  c = zeros(1,N, 'like', sum);
end
[O,s,C]=  cahaos1(x, sum,y,c);
y=O;
sum=s;
c=C;
Out=O(1);
end
```

Fig. 6 MATLAB script for Cahaos function



```
clc
clear
Nc=2;
mu=0.01;
deg=512;
y1= zeros(deg);
y2= zeros(deg);
z= zeros(deg);
nshift = 4;
nsamp = 100; %number of samples
nbits = 32; % fixed-point word length
nfrac = 31; % fixed-point fraction length
for i=1:deg
 x=fi(rand(1,1), numerictype(0,nbits,nfrac));%rand(1);
[O]=  cahaos(x);
y(i)=O;
end
t=1:deg;
figure(1)
plot(y);
title('Figure-Adaptive-Digital Realization of Chaos,y(k),c1=c2=c3=0.0');
```

Fig. 7 Cahaos Tester Code

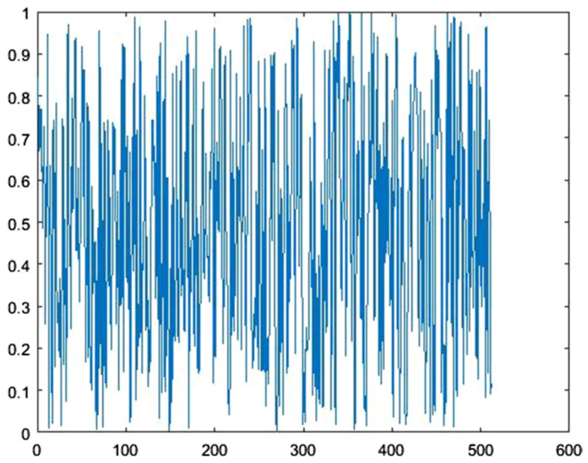


Fig. 8 The plot of the output of the Adaptive chaotic random number generator

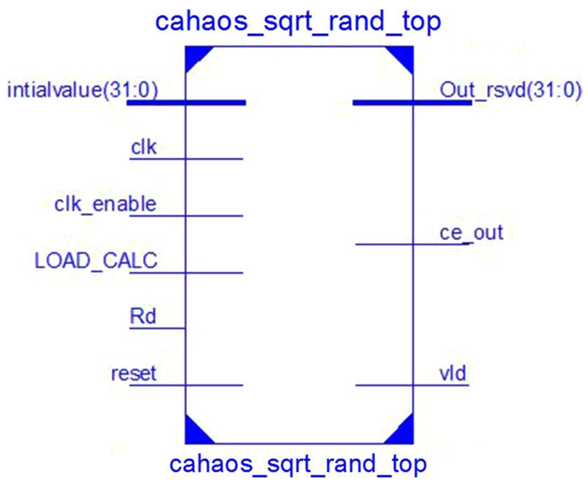


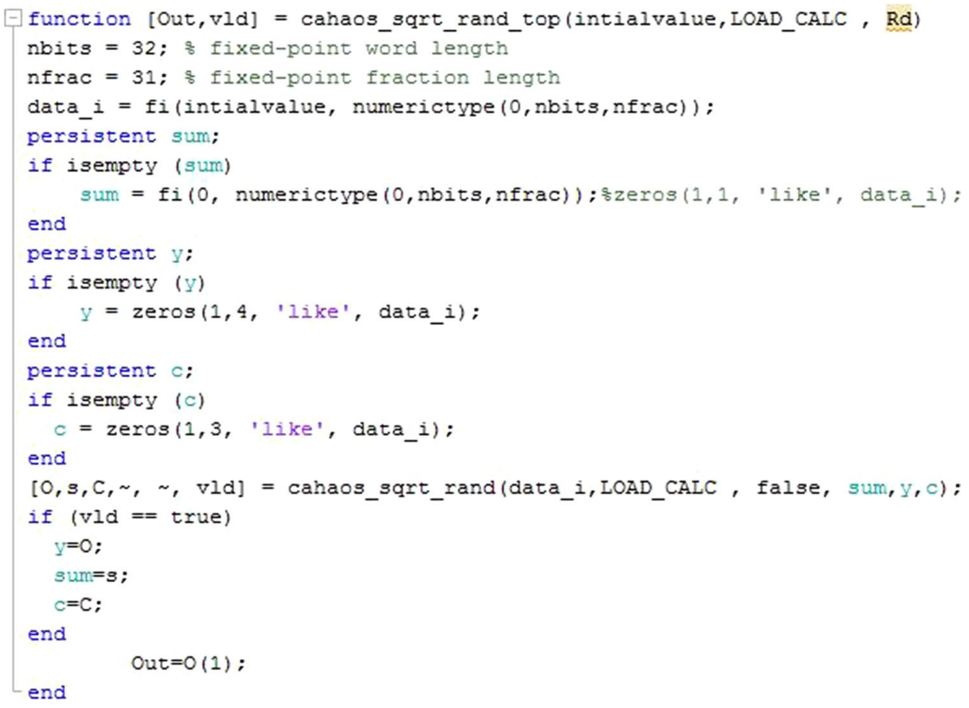
Fig. 9 Top level of the chaotic random number key grnerstor based on swuare root PRNG seed

paper, we propose an efficient and configurable FPGA Design for adaptive digital chaotic key generator. Our design has another advantage of storing the output keys internally and reading them later.

Discussion
The usage of the Chaotic is very wide. In Kumar et al. (2023) A combined chaotic key generator (CCKG) is proposed to enhance key sensitivity and generation to improve the security of medical images to be encrypted. In Kiran and Parameshachari (2022) they combine the image encryption and chaotic system to improve the security. In Sha et al. (2023) the paper proposed an image encryption algorithm using a hybrid of three modified and improved chaotic one-dimensional (1D) maps to avoid the shortcomings of 1D maps and multidimensional (MD) maps. In Zhao et al. (2023) they enhanced the randomness of the constructed chaotic system and expand its key space. In Yildirim and Tanyildizi (2023) The Mode2(x) method and the unpredictable nature of chaotic systems are used for random number generation. In Bonny et al. (2023) The paper presented a new, highly secure chaos-based secure communication system that combines a conventional cryptography algorithm with two levels of chaotic masking technique. In Al-Saadi and Alshawi (2023) they introduce Light Encryption Device (LED) which is a high-performance, lightweight block encryption solution that works on resource-constrained devices and considers a lighter version of AES. They provided Chaos-based encryption as an exceptionally high level of security because of the unique characteristics of chaotic systems. In Bonilla et al. (2016) they introduced a true random number generator based on chaos.

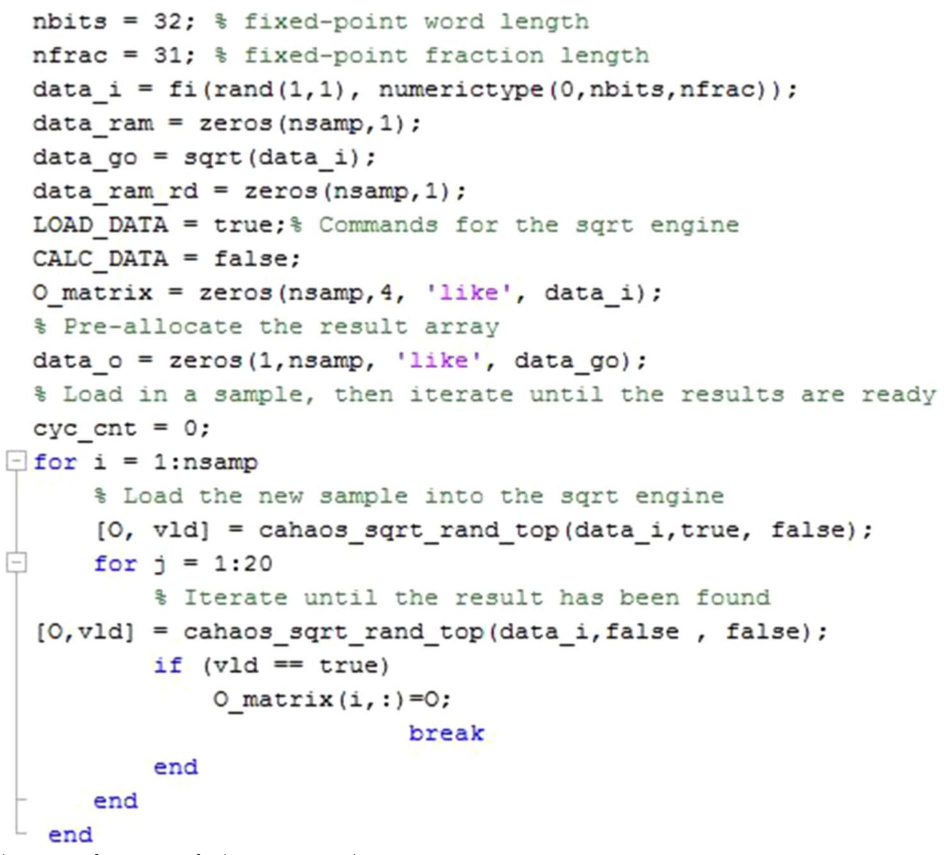
Many PRNGs are implemented as an example of FPGA implementation of PRNG (Rezk et al. 2019). They also adopted an already published and tested algorithm and implemented it on the FPGA (XC5VLX50T), a large FPGA compared to ours. Their implementation has utilized 1.4% and 16.7% of the available slices and DSP blocks respectively. this is huge utilization compared to ours The Maximum frequency was 78which which is much faster than ours. They didn't use MATLAB HDL Coder. The most usage of the PRNG is for key generation, key generation over FPGA can be found in Srinivas and Janaki (2021) and Naiem et al. (2009). Many other implementations of chaotic over FPGA for many applications like Abd El-Maksoud et al. (2020), Tutueva et al. (2020), Al-Musawi et al. (2021) and Dridi et al. (2020).

Conclusions
In this paper, we introduced the design and implementation of an adaptive chaotic key generator over FPGA. The design and its implementation were done by using MATLAB HDL Coder. The target FPGA was Xilinx Spartan 3 xc6slx9-2-cpg196. We introduced two designs; design 1 was the implementation of the algorithm with a random seed input. and design 2 was the implementation of the algorithm with input non-square number and the random seed is internally generated. Design 2 also has memory to store the generated output that can be accessed later. For design 1, the area utilization was for design 1, 189 slice registers, 2303 slice LUTs (lookup tables), 126 fully used LUT, 68 IOs, and 16 DSP blocks. is the maximum frequency was 15.711 MHz. While for design 2, the area utilization was 304 slice registers, 3380 slice LUTs (lookup tables), 276 fully used LUT, 70 IOs, and 16 DSP blocks. is the maximum frequency is 11.635 MHz.



```
function [Out,vld] = cahaos_sqrt_rand_top(intialvalue,LOAD_CALC , Rd)
nbits = 32; % fixed-point word length
nfrac = 31; % fixed-point fraction length
data_i = fi(intialvalue, numerictype(0,nbits,nfrac));
persistent sum;
if isempty (sum)
    sum = fi(0, numerictype(0,nbits,nfrac));%zeros(1,1, 'like', data_i);
end
persistent y;
if isempty (y)
    y = zeros(1,4, 'like', data_i);
end
persistent c;
if isempty (c)
  c = zeros(1,3, 'like', data_i);
end
[O,s,C,~, ~, vld] = cahaos_sqrt_rand(data_i,LOAD_CALC , false, sum,y,c);
if (vld == true)
  y=O;
  sum=s;
  c=C;
end
        Out=O(1);
end
```

Fig. 10 Script of cahaos_sqrt_rand_top



```
 nbits = 32; % fixed-point word length
 nfrac = 31; % fixed-point fraction length
 data_i = fi(rand(1,1), numerictype(0,nbits,nfrac));
 data_ram = zeros(nsamp,1);
 data_go = sqrt(data_i);
 data_ram_rd = zeros(nsamp,1);
 LOAD_DATA = true;% Commands for the sqrt engine
 CALC_DATA = false;
 O_matrix = zeros(nsamp,4, 'like', data_i);
 % Pre-allocate the result array
 data_o = zeros(1,nsamp, 'like', data_go);
 % Load in a sample, then iterate until the results are ready
 cyc_cnt = 0;
 for i = 1:nsamp
     % Load the new sample into the sqrt engine
     [O, vld] = cahaos_sqrt_rand_top(data_i,true, false);
     for j = 1:20
         % Iterate until the result has been found
 [O,vld] = cahaos_sqrt_rand_top(data_i,false , false);
         if (vld == true)
             O_matrix(i,:)=O;
                         break
         end
     end
  end
```

Fig. 11 Code for the tester of Top view of cahaos_sqrt_rand_top

Table 1 The estimated values for device utilization for Design1

Logic utilization	Used	Available	Utilization (%)
Number of Slice Registers	189	11440	1
Number of Slice LUTs	2303	5720	40
Number of fully used LUT-FF pairs	126	2366	5
Number of bonded IOBs	68	106	64
Number of BUFG/BUFGCTRL/BUFHCEs	1	16	6
Number of DSP48A1s	16	16	100

Table 2 The estimated values for device utilization for design 2

Logic utilization	Used	Available	Utilization (%)
Number of Slice Registers	304	11440	2
Number of Slice LUTs	3380	5720	59
Number of fully used LUT-FF pairs	276	3408	8
Number of bonded IOBs	70	106	66
Number of BUFG/BUFGCTRL/BUFHCEs	1	16	6
Number of DSP48A1s	16	16	100

Abbreviations

AES	Advanced Encryption Standard
FPGA	Field programmable gate array
LUT	Look up table
HDL	Hardware description language

Acknowledgements
The authors would like to acknowledge Electronics Research Institute (ERI), Egypt for supporting us with the simulation tool (MATLAB).

Author contributions
GE, ES and SK have contributed equally to the manuscript. They have read and approved the manuscript.

Authors' Information
Ghada Elsayed has received her Bsc, Msc, and PhD degrees in electronic and communication engineering, 2001, 2005, and 2008 respectively. She worked for the Egyptian Space Program for the first seven years then she sifted to academic career, in which she worked for more than five years for MSA University then she travelled as a visitor researcher to Japan, Kyushu Institute of Technology. After that, she continued for the same path in MTI University. In 2014, Ghada published a book about securing satellites control link. She also published many scientific papers.
Elsayed Soleit, Electrical Engineering Dept. Faculty of Engineering, The Modern University For Technology and Information. Professor of digital signal processing and data communication since 2000. Ph.D. in Electronic Engineering, Information Technology Faculty, University of Kent, United Kingdom, UK. March 1989. The area of interest and research are adaptive signal processing, data communication, computer networks, encryption/decryption algorithms, and image processing.
Somaya Kayed is an Associate Professor and a head of Electrical Dept. (Electronics, communication, computer and control Engineering) at Oubor Higher institute for Engineering and technology. She is graduated in 1987 from Ain Shams University with a BSc. in Electronics and Communications department, with general grade (very Good) 80% her order of merit 13 of the successful students totaling (75). Her graduation project tackled Distinction and she was top ranked as the 13th on her class. In 1997, she finished her Masters of Science (MSc.) in the same department. Afterwards, in 2000, she awarded her PhD also from Ain Shams University under the supervision of Prof. Hani Fikry Ragaie. She was an acting dean for the 2019 first term at Oubor Higher institute for Engineering and technology, she has published in local and international conference many scientific papers as well as multiple books related to her research interests (Analog and digital VLSI design, current conveyor, Nano electronics). At the same time, her passion for development is not limited to academia. She is also a volunteer member in NGO. On one side, these diverse experiences enriched not only her team work skills but also her leadership competencies.

Funding
No funding was obtained for this study

Availability of data and materials
We provide the code and the results in this manuscript. any required data or materials are available if needed.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 16 March 2023 Accepted: 1 August 2023
Published online: 11 August 2023

References
Abd El-Maksoud AJ, Abd El-Kader AA, Hassan BG, Rihan NG, Tolba MF, Said LA, Radwan AG, Abu-Elyazeed MF (2020) FPGA implementation of integer/fractional chaotic systems. Multimedia security using chaotic maps: principles and methodologies, pp 199–229
Al-Musawi WA, Wali W, Al-Ibadi MA (2021) Implementation of chaotic system using FPGA. In: 2021 6th Asia-Pacific conference on intelligent robot systems (ACIRS), IEEE, pp 1–6
Al-Saadi HM, Alshawi I (2023) Provably-secure led block cipher diffusion and confusion based on chaotic maps. Informatica 47(6)
Bonilla LL, Alvaro M, Carretero M (2016) Chaos-based true random number generators. J Math Ind 7:1–17
Bonny T, Nassan WA, Baba A (2023) Voice encryption using a unified hyper-chaotic system. Multimed Tools Appl 82(1):1067–1085
Digilent (2023) Cmod S6 FPGA Board Reference Manual. https://digilent.com/reference/_media/reference/programmable-logic/cmod-s6/cmods6_rm.pdf. Accessed 20
Dridi F, Atamech C, El Assad S, Youssef WE, Machhout M (2020) FPGA implementation of a chaos-based stream cipher and evaluation of its performances. Int J Chaotic Comput 7(1):179–186
Eda techchannel OpenSystems Media: MathWorks Introduces HDL Coder and Verifier For MATLAB (2012). http://tech.opensystemsmedia.com/eda/2012/03/mathworks-introduces-hdl-coder-and-verifier-for-matlab/. Accessed 20 (2023)
Elsayed G, Kayed SI (2022) A comparative study between MATLAB HDL Coder and VHDL for FPGAs design and implementation. J Int Soc Sci Eng 4:92–98
Kayed SI, Elsayed G (2022) Optimizing Techniques for using MATLAB HDL Coder. https://aeas2022.asu.edu.eg
Kiran P, Parameshachari B (2022) Resource optimized selective image encryption of medical images using multiple chaotic systems. Microprocess Microsyst 91:104546
Kumar D, Sudha V, Ranjithkumar R (2023) A one-round medical image encryption algorithm based on a combined chaotic key generator. Med Biol Eng Comput 61(1):205–227
MathWorks, inc: HDL Coder^{TM} User's Guide© COPYRIGHT 2012-2015 (2012). https://www.mathworks.com/help/hdlcoder/. Accessed 20 (2023)

Naiem GF, Elramly S, Hasan BEM, Shehata K (2009) New symmetric key generation algorithm. In: 2009 national radio science conference, IEEE, pp 1–8
Rezk AA, Madian AH, Radwan AG, Soliman AM (2019) Reconfigurable chaotic pseudo random number generator based on FPGA. AEU-Int J Electr Commun 98:174–180
Sha Y, Mou J, Wang J, Banerjee S, Sun B (2023) Chaotic image encryption with hopfield neural network. Fractals 2340107
Soleit EA (2018) A new adaptive chaotic key generator. In: The international conference on electrical engineering, Military Technical College, vol 11, pp 1–8
Srinivas K, Janaki V (2021) Symmetric key generation algorithm using image-based chaos logistic maps. Int J Adv Intell Paradigms 19(3–4):393–409
Su J, McSweeney J (2019) Square root pseudo-random number generators. PhD thesis, Rose-Hulman Institute of Technology, Mathematics Department
Tutueva AV, Nepomuceno EG, Karimov AI, Andreev VS, Butusov DN (2020) Adaptive chaotic maps and their application to pseudo-random numbers generation. Chaos Solitons Fractals 133:109615
Yildirim G, Tanyildizi E (2023) An innovative approach based on optimization for the determination of initial conditions of continuous-time chaotic system as a random number generator. Chaos Solitons Fractals 172:113548
Zhao W, Chang Z, Ma C, Shen Z (2023) A pseudorandom number generator based on the chaotic map and quantum random walks. Entropy 25(1):166

Publisher's Note