Hardware Implementation of Binary Kolakoski Sequence

Rabie A. Mahmoud
Computer Engineering Department,
Arab Academy for Science, Technology & Maritime Transport, Latakia Branch,
Latakia, Syria.
rabiemah@yahoo.com

Abstract: Kolakoski Unit (KU) is a simple structure designed to implement the binary form of Dekking substitutions which are used to generate the binary form of the infinite Kolakoski sequence. This Kolakoski Unit uses one OR gate, one AND gate, and two Tri-state buffers. In addition, we provide Field Programmable Gate Array (FPGA) implementation of generating the first 128-bit of binary Kolakoski sequence.

Keywords: Binary Kolakoski Sequence, PRNG, Cryptography, FPGA.

1. Introduction

Generalized Kolakoski sequence which is a cube-free infinite self-describing sequence over the alphabet {1, 2} can be converted into binary form by converting 1's into 0's and 2's into 1's. The binary Kolakoski sequence has the same properties of generalized Kolakoski sequence which can be useful for cryptography like using in stream cipher or designing a pseudo random number generator. The three major properties of Kolakoski sequence are aperiodic sequence, a cube-free sequence which means that does not contain three consecutive identical words where the word may be one or more letters, and finally the density of letters in the sequence are limited to 0.5. The Kolakoski Unit (KU) is considered the engine of binary Kolakoski sequence generator where one OR gate, one AND gate, and two Tri-state buffers are used to perform binary Dekking substitutions to generate binary Kolakoski sequence. In the following sections, we provide a brief discussion about generalized Kolakoski sequence, the structure of Kolakoski Unit, the formal description of binary Kolakoski Sequence generator algorithm, and FPGA implementations of generating the first 128-bit of binary Kolakoski sequence, then a summary and our conclusions.

2. Kolakoski Sequence

Kolakoski sequence [1] proposed by William Kolakoski in 1965 but it was considered earlier by Rufus Oldenburger in 1939 [2]. Kolakoski sequence K is a cube-free infinite sequence ("self-describing" sequence) over the alphabet A={1, 2}, which starts with 1 and equals the sequence defined by its run lengths:

K= 1 2 2 1 1 2 1 2 2 1 2 2 1 1 2 1 1 2 2 1 …
1 2 2 1 1 2 1 2 2 1 2 … = K

A run is a maximal subsequence of consecutive identical symbols [3]. Kolakoski sequence can be generated by starting with 2 as a seed and iterating the two substitutions σ_0 and σ_1 alternatingly where

$$\sigma_0: \begin{matrix} 1 \mapsto 2 \\ 2 \mapsto 22 \end{matrix} \quad \text{and} \quad \sigma_1: \begin{matrix} 1 \mapsto 1 \\ 2 \mapsto 11, \end{matrix}$$

σ_0 substitutes letters on even positions and σ_1 substitutes letters on odd positions:

$2 \mapsto 22 \mapsto 2211 \mapsto 221121 \mapsto 221121221 \mapsto \dots$.

Figure1 shows the generalized sequential machine (GSM) of infinite sequence of Kolakoski by iteration starting with the letter 2 [4].

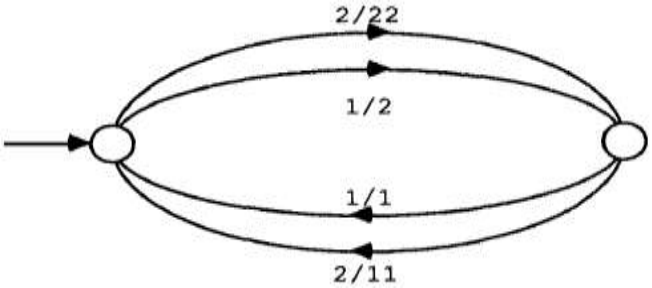

Figure 1. Kolakoski's Generalized Sequential Machine

Also, Kolakoski sequence can be generated by the initial word 22 and the substitutions which are defined by Dekking [5] as:

$$\begin{matrix} 11 \mapsto 21 \\ 22 \mapsto 2211 \end{matrix} \qquad \begin{matrix} 12 \mapsto 211 \\ 21 \mapsto 221. \end{matrix}$$

Figure2 shows the first five Dekking substitutions of reading and writing Kolakoski letters starting with word 22 to generate Kolakoski sequence.

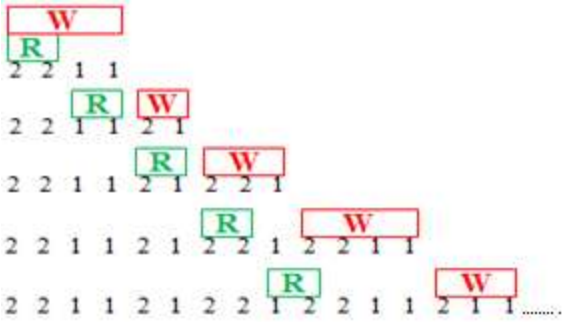

Figure 2. Generalized Kolakoski by Dekking Substitutions

More details of Kolakoski sequences, the connection between

generalized Kolakoski sequences and model sets can be found in [6], [7], [8], and [9]. Related problems of Kolakoski sequence and the proof of Kolakoski sequence which are not eventually periodic can be found in [10]. Although the conjecture of Kolakoski sequence remains unproved, but it seems plausible that the upper density of 1's as well as the upper density of 2's is 0.5. A clever approach by Chvátal [11] proved that density of letters of Kolakoski sequence is limited to 0.5±0.000838.

3. Kolakoski Unit Structure

A binary form of Kolakoski sequence through decimal expansion of Kolakoski constant [12] can be obtained by arbitrarily converting 1's into 0's and 2's into 1's:

K= 0 1 1 0 0 1 0 1 1 0 1 1 0 0 1 0 0 1 1 0 … .

Also, binary Kolakoski sequence can be obtained by converting the Dekking substitutions into binary form as:

00 ↦ 10 01 ↦ 100

11 ↦ 1100 10 ↦ 110.

Figure3 shows generalization of binary Kolakoski sequence by using the binary form of Dekking substitutions and starting with word 11.

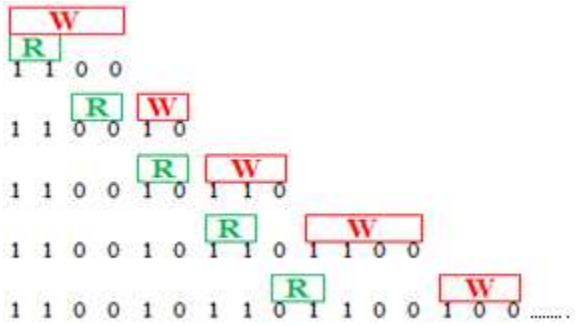

Figure 3. Binary Kolakoski by Binary Dekking Substitutions

Kolakoski Unit (KU) is designed to implement the binary form of Dekking substitutions to generate binary Kolakoski sequence. The basic Kolakoski Unit is shown in Figure4. The basic Kolakoski Unit is performed by OR gate, AND gate, and two Tri-state buffers. The inputs of Kolakoski Unit is just two bits where "00" input produces "ZZ10" as output, "01" input produces "Z100" as output, "10" input produces "Z110" as output, and "11" input produces "1100" as output where "Z" is high-impedance state which means no current in the bus. In other words, the outputs of Kolakoski Unit are "10", "100", "110", and "1100" for "00", "01", "10", and "11" respectively. Table1 shows the truth table of Kolakoski unit.

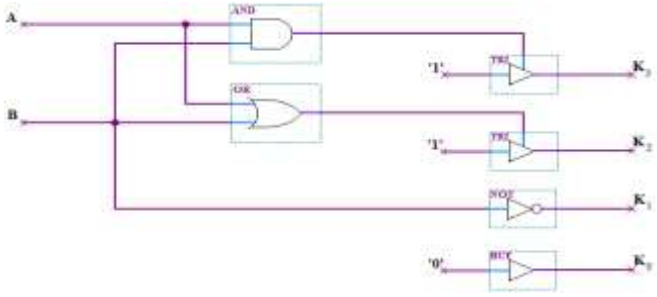

Figure 4. Structure of Kolakoski Unit

Table1: Kolakoski Unit Truth Table

A	B	KU			
		K_3	K_2	K_1	K_0
0	0	hi-Z*	hi-Z	1	0
0	1	hi-Z	1	0	0
1	0	hi-Z	1	1	0
1	1	1	1	0	0

* Hi-impedance State

4. The Algorithm

In this section, we provide the formal description of the binary Kolakoski sequence generator as follows:

Algorithm: BINARY KOLAKOSKI GENERATOR
Output: *Binary-Kolakoski-Sequence K*
Algorithm body:
Begin
Begin Generation
1. *Store '0' in the first bit of Binary-Kolakoski-Sequence K;*
2. *Read a pair of consecutive bits P="11" into the pair cache as initial pair for binary Kolakoski generator;*
3. *Generate new-bits for Binary-Kolakoski-Sequence by applying Kolakoski Unit function on the pair P;*
4. *Load the next pair of consecutive bits P from Binary-Kolakoski-Sequence into the pair cache;*
5. *Repeat steps 3 to 4 if Binary-Kolakoski-Sequence cache is not empty;*
6. *When the Binary-Kolakoski-Sequence cache is finished then halts;*
End Generation;
End Algorithm.

Function Kolakoski Unit
Begin
1. *Read the pair bits;*
2. *Process the substitution on the pair bits as:*
2.1 *If the pair is "00" then produce the output as "10",*
2.2 *If the pair is "01" then produce the output as "100",*
2.3 *If the pair is "10" then produce the output as "110",*
2.4 *If the pair is "11" then produce the output as "1100";*
3. *Store the producing output at the end of Binary-Kolakoski-Sequence K;*
End;

5. Hardware Implementation

The simple structure of Kolakoski Unit (KU) which is the corner stone of binary Kolakoski generator and the important properties of Kolakoski sequence leads to FPGA-based implementation. We have implemented the binary Kolakoski generator applying the VHDL hardware description language and utilizing Altera design environment Quartus II 13.0 Service Pack 1 Web Edition [13] with ModelSim Altera Starter Edition 10.1d [14]. Kolakoski Unit consumes just 2 logic elements in any FPGA chip to perform one OR gate, one AND gate, and two Tri-state buffers but FPGA synthesis of generating the first 128-bit of binary Kolakoski sequence consumed 2696 logic elements cause the algorithm of generating Kolakoski sequence by reading the previous inputs of Kolakoski unit then writing the output at the end of generated sequence in previous clock consumes a lot of logic elements to perform the final register which contains the desired binary Kolakoski sequence. So, generating more bits

of binary Kolakoski sequence needs more logic elements and memory in the device. The FPGA design was implemented using EP4CGX150DF31I7AD, Cyclone IV GX family device. The implementation results and the schematic diagram of generating the first 128-bit of binary Kolakoski sequence are shown in Figure5. The technology map viewer and RTL screen for Kolakoski unit are shown in Figures 6, and 7 respectively. Also, the technology map viewer and RTL screens of generating the first 128-bit of binary Kolakoski sequence are shown in Figures 8, 9, and 10 respectively. Figure 11 displays the Kolakoski generator simulation showing the final output of the first 128-bit of binary Kolakoski sequence, and showing the input and output of Kolakoski unit at each clock of device. Figure12 demonstrates the floor plan for generating the first 128-bit of binary Kolakoski sequence. The details of the analysis and synthesis summary and timing analyzer are shown in appendix A. Appendix B represents sample VHDL code for Kolakoski Unit and binary Kolakoski Sequence.

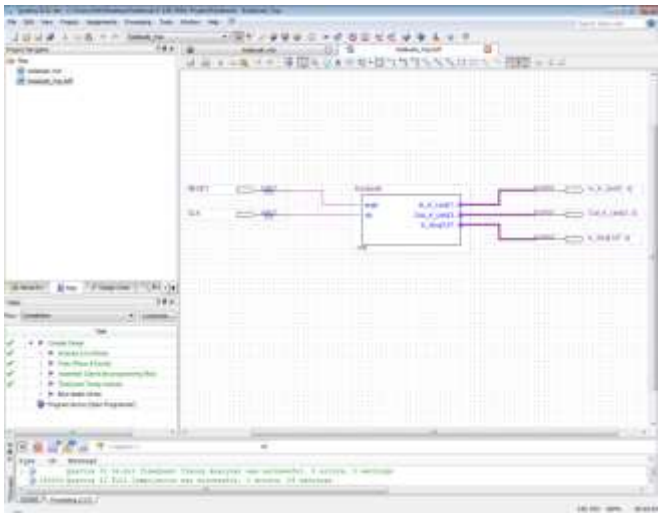

Figure 5. Schematic diagram of generating the first 128-bit of binary Kolakoski sequence showing also correct implementation

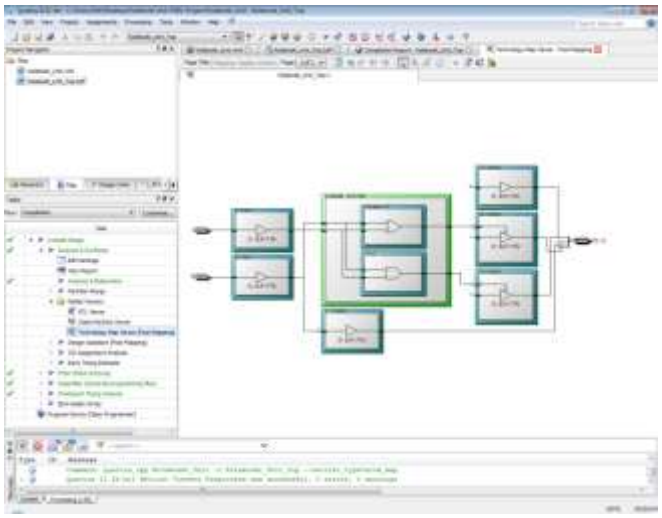

Figure 6. Technology Map Viewer of Kolakoski Unit

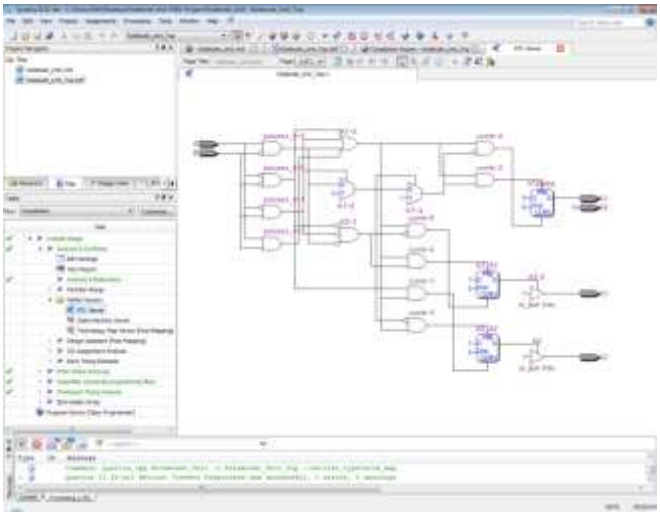

Figure 7. RTL screen for Kolakoski Unit

Figure 8. Technology Map Viewer of generating the first 128-bit of binary Kolakoski sequence

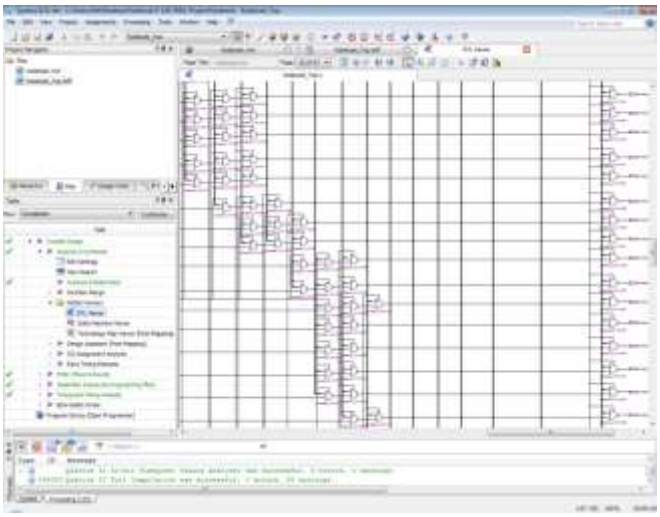

Figure 9. RTL screen for part of generating the first 128-bit of binary Kolakoski sequence implementation

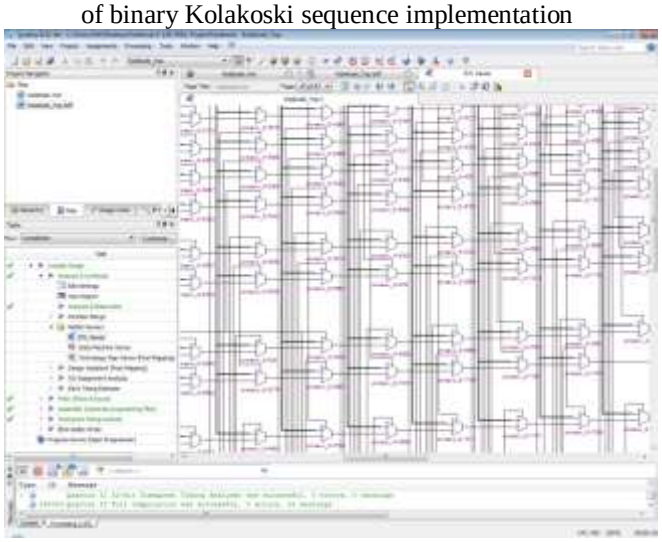

Figure 10. RTL screen for part of generating the first 128-bit of binary Kolakoski sequence implementation

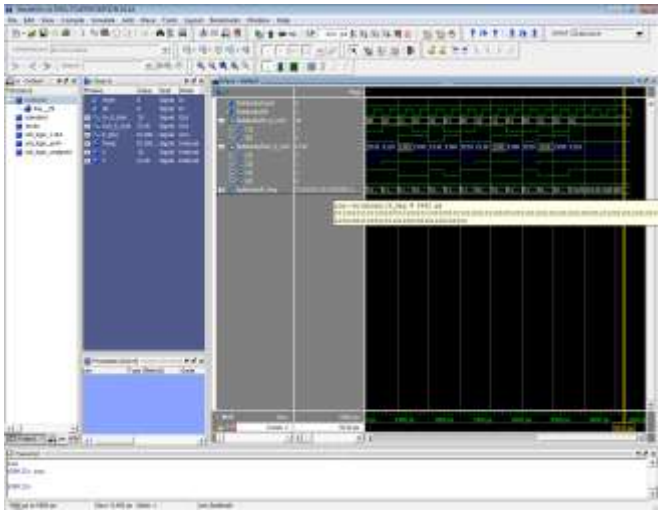

Figure 11. ModelSim simulator screen showing the first 128-bit of binary Kolakoski sequence as output

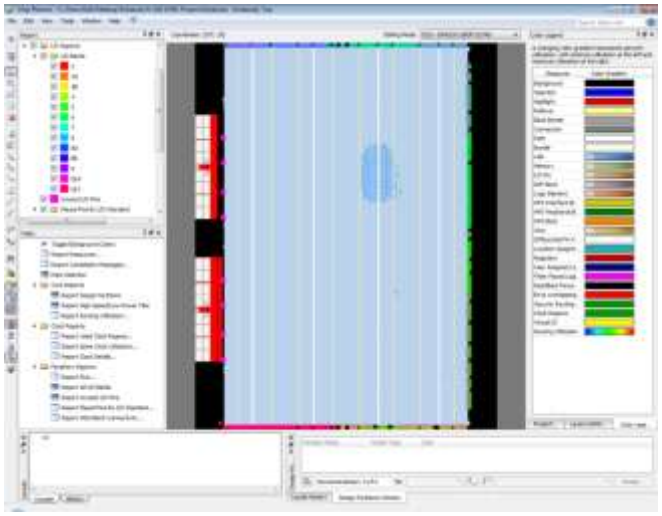

Figure 12. Floor-plan of chip for generating the first 128-bit of binary Kolakoski sequence

6. Summary and Conclusions

Application like a binary sequence generator based on Kolakoski sequence [15] led us to implement binary Kolakoski sequence where the method of adding disorder Kolakoski bits to the original sequence which produced from pseudo or even true random number generator can be increase the entropy of the generator through merging the properties of Kolakoski sequence in the original output of generator. Practically, a simple architecture based on meta-stable circuit [16] by putting together the binary sequence generator and binary Kolakoski generator in a configuration at two different frequencies will merge the binary Kolakoski sequence with any generated binary sequence in disorder way where a high speed of generating Kolakoski bits is required to use and spread a maximum number of Kolakoski bits in the original binary sequence. We have designed a Kolakoski Unit to be a block of generating binary Kolakoski sequence although the substitutions of generating binary Kolakoski sequence are not morphic through simplest way by using one OR gate, one AND gate, and two Tri-state buffers. Also, we have provided a proof-of-concept FPGA implementation by generating the first 128-bit of binary Kolakoski sequence. A comparison with other implementations is not applicable since this is the first time the Kolakoski is FPGA implemented. This and other related issues as other applications which used Kolakoski sequence will be dealt with in future. Moreover, we noticed that:

The binary Kolakoski sequence [17] is obtained by using our design of Kolakoski Unit starting with 11, the binary Kolakoski sequence without the first 2-bit which are 11 is obtained by using our design starting with 00, and the binary Kolakoski sequence without the first 4-bit which are 1100 is obtained by using our design starting with 10, but a new sequence is obtained by using our design of Kolakoski Unit starting with 01

01 ↦
01100 ↦
01100110 ↦
01100110100 ↦
01100110100110 ↦
01100110100110110 ↦
011001101001101101001 ↦
0110011010011011010011011100 ↦
01100110100110110100110110010010 ↦
011001101001101101001101100100101100 ↦
0110011010011011010011011001001011001100 ↦
01100110100110110100110110010010110010011 0 ↦ . . .

In other hand, we can be generate this new sequence over alphabet {1, 2} as
122112211211221221211221221121121221121122 1 . . .
Still, the questions are:

Is this new sequence a part form generalized Kolakoski sequence or not?

If not, did this new sequence have the same properties of generalized Kolakoski sequence?

References

[1] William Kolakoski, "Self Generating Runs, Problem 5304," American Mathematical Monthly, Vol.72, p.674, 1965.
Solution by Necdet Üçoluk in: American Mathematical Monthly, Vol.73, pp. 681-682, 1966.

[2] Rufus Oldenburger, "Exponent Trajectories in Symbolic Dynamics," Transactions of the American Mathematical Society, Vol.46, pp.453-466, 1939.

[3] Neil James Alexander Sloane, The On-line Encyclopedia of Integer Sequences site: https://oeis.org/A000002

[4] Jean Berstel, "Properties of Infinite Words: Recent Results," Lecture Notes in Computer Science, STACS 89, 6th Annual Symposium on Theoretical Aspects of Computer Science, Paderborn, Germany, Vol.349, pp.36-46, Feb., 1989.

[5] Frederik M. Dekking, "Regularity and Irregularity of Sequences Generated by Automata," Séminaire de Théorie des Nombres de Bordeaux, 1979-1980, exposé No.9, pp.1-10, Nov., 1979.

[6] Bernd Sing, "Kolakoski-(2*m*,2*n*) are limit-periodic model sets," Journal of Mathematical Physics, Vol.44, Issue 2, pp.899-912, Feb. 2003.

[7] Bernd Sing, "Kolakoski sequences - an example of aperiodic order," Journal of Non-Crystalline Solids, Vol.334-335, pp.100-104, Mar., 2004.

[8] Michael Baake, and Bernd Sing, "Kolakoski-(3,1) is a (Deformed) Model Set," Canadian Mathematical Bulletin, Vol.47, No.2, pp.168-190, Jun., 2004.

[9] Olivier Bordellès, and Benoit Cloitre, "Bounds for the Kolakoski Sequence," Journal of Integer Sequences, Vol.14, Article 11.2.1, 2011.

[10] Bernd Sing, "More Kolakoski Sequences," INTEGERS Electronic Journal of Combinatorial Number Theory, Vol.11B, 2011. Proceedings of the Leiden Numeration Conference 2010, Lorentz Center of Leiden University, Netherlands.

[11] Vašek Chvátal, "Notes on the Kolakoski Sequence," DIMACS Technical Report 93-84, Dec, 1993.

[12] Neil James Alexander Sloane, The On-line Encyclopedia of Integer Sequences site: http://oeis.org/A118270

[13] Altera's user-support site: http://www.altera.com/support/examples/vhdl/vhdl.html

[14] ModelSim-Altera software support site: http://www.altera.com/support/software/products/modelsim/mod-modelsim.html

[15] Joel Reyes Noche, "A Binary Sequence Generator Based on the Kolakoski Sequence and Multiples of Odd Primes," Kamawotan, Vol.1, No.2, pp.79-87, May, 2007.

[16] Çetin Kaya Koç, "Cryptographic Engineering," Springer, 2009.

[17] Neil James Alexander Sloane, The On-line Encyclopedia of Integer Sequences site: http://oeis.org/A157686

Appendix A: The analysis & synthesis and timing report details

The usage number of logic elements and their connections in the device can be changed depending on the optimization technique which is used for synthezing. Table2 and Table3 show the number of usage logic elements and the interconnections of generating the first 128-bit of Kolakoski sequence between them in Balanced, Area, and Speed optimization technique. Also, the comparison between optimization techniques was extracted from the timing reports of implementing Balanced, Area, and Speed optimization. Table4 shows a timing comparison between various implementation delays.

Fitter Resource Usage and Routing Usage Summary

- Family: Cyclone IV GX
- Device: EP4CGX150DF31I7AD
- Nominal Core Voltage: 1.20 V
- Minimum Core Junction Temperature: -40 °C
- Maximum Core Junction Temperature: 100 °C.
- Optimization Technique: Balanced
- Total logic elements: 2,696 out of 149,760 (2 %)
  - -- Combinational with no register: 2,565
  - -- Register only: 1
  - -- Combinational with a register: 130
- Logic element usage by number of LUT inputs
  - -- 4 input functions: 2,106
  - -- 3 input functions: 288
  - -- <=2 input functions: 301
  - -- Register only: 1
- Logic elements by mode
  - -- Normal mode: 2,690
  - -- Arithmetic mode: 5
- Total registers: 131 out of 152,165 (< 1 %)
  - -- Dedicated logic registers: 131 out of 149,760
  - -- I/O registers:0 out of 2,405
- Total LABs: 182 out of 9,360 (2 %)
- Maximum fan-out: 181
- Highest non-global fan-out: 181
- Total fan-out: 10,547
- Average fan-out: 3.39

- Block interconnects: 5,089 out of 445,464 (1 %)
- C16 interconnects: 229 out of 12,402 (2 %)
- C4 interconnects: 3,439 out of 263,952 (1 %)
- Direct links: 196 out of 445,464 (< 1 %)
- Global clocks: 1 out of 30 (3 %)
- Local interconnects: 1,955 out of 149,760 (1 %)
- R24 interconnects: 215 out of 12,690 (2 %)
- R4 interconnects: 4,058 out of 370,260 (1 %)

Table2. A resource usage comparison between optimization technique implementations of generating the first 128-bit of Kolakoski Sequence

	Balanced	Area	Speed
Total logic elements	2696	2685	2768
Combinational with no register	2565	2554	2637
Register only	1	0	0
Combinational with a register	130	131	131
Total combinational functions	2696	2685	2768
4 input functions	2106	2124	2241
3 input functions	288	302	198
<=2 input functions	301	259	329

Register only	1	0	0
Total fan-out	10547	10576	10872
Maximum fan-out	181	170	240
Average fan-out	3.39	3.41	3.42

Table3. A routing usage comparison between optimization technique implementations of generating the first 128-bit of Kolakoski Sequence

	Balanced	Area	Speed
Block interconnects	5089	4964	5323
C16 interconnects	229	318	309
C4 interconnects	3439	3641	3840
Direct links	196	210	209
Global clocks	1	1	1
Local interconnects	1955	1994	1988
R24 interconnects	215	175	226
R4 interconnects	4058	3915	4339

Timing Analyzer Summary

• Longest propagation delay RR which is measured from rising edge to rising edge was 14.566 ns and longest delay FF which is measured from falling edge to falling edge was 15.231 ns from input port "RESET" to output port "K_Seq[82]". While the RR and FF for longest minimum propagation delay were 7.065 ns and 8.069 ns respectively.

• Longest clock-to-output from clock port "CLK" to data port "K_Seq[82]" was 14.980 ns and 15.163 ns for rise and fall clock edge respectively. While the longest minimum clock-to-output was 7.747 ns and 7.836 ns for rise and fall clock edge respectively.

• Setup time (t_{SU}) which is report to the setup times for clocks in the design was 7.243 ns and 7.915 ns for rise and fall clock edge respectively for data port "RESET" and clock port "CLK".

• Hold time (t_H) which is reports to the hold times for clocks in the design was -0.885 ns and -1.591 ns for rise and fall clock edge respectively for data port "RESET" and clock port "CLK".

Table4. A timing comparison between optimization technique implementations of generating the first 128-bit of Kolakoski Sequence

		Balanced	Area	Speed
Propagation delay	RR	14.566	15.616	16.128
	FF	15.231	16.296	16.643
Minimum propagation delay	RR	7.065	7.634	7.688
	FF	8.069	8.696	8.760
Clock-to-output	Rise	14.980	15.089	15.013
	Fall	15.163	14.996	15.002
Minimum clock-to-output	Rise	7.747	7.604	7.727
	Fall	7.836	7.684	7.911
Setup time	Rise	7.243	7.502	8.276
	Fall	7.915	8.169	8.745
Hold time	Rise	-0.885	-1.211	-1.128
	Fall	-1.591	-1.953	-1.876

Appendix B:

B.1. Sample VHDL code for Kolakoski_Unit

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Kolakoski_Unit IS
    PORT(A : in std_logic;
         B : in std_logic;
         K : out std_logic_vector(3 downto 0));
END Kolakoski_Unit;

ARCHITECTURE behavioral OF Kolakoski_Unit IS
BEGIN
  Process(A, B)
  Begin
    If   A='0' AND B='0'  then K <= "ZZ10";
    elsif A='0' AND B='1'  then K <= "Z100";
    elsif A='1' AND B='0'  then K <= "Z110";
    elsif A='1' AND B='1'  then K <= "1100";
    End if;
  End Process;
END behavioral;
```

B.2. Sample VHDL code for generating the first 128-bit of Kolakoski_Sequence

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Kolakoski_Sequence IS
    PORT(reset : in  std_logic;
         clk   : in  std_logic;
         In_K_Unit  : out std_logic_vector(1  downto 0);
         Out_K_Unit: out std_logic_vector(3  downto 0);
         K_Seq   : out std_logic_vector(127 downto 0));
END Kolakoski_Sequence;

ARCHITECTURE behavioral OF Kolakoski_Sequence IS
 Signal Temp:std_logic_vector(127downto0):=(others=>'1');
 Signal X : std_logic_vector(1 downto 0);
 Signal Y : std_logic_vector(3 downto 0);
BEGIN
  Process(reset, clk, Temp)
      Variable  i  :  natural range 126 downto 0 := 126;
  Begin
   If (reset ='1') then K_Seq       <= (others => '1');
                        In_K_Unit   <= (others => '1');
                        Out_K_Unit <= (others => '1');    else

    Temp(127) <= '0';
    Temp(126 downto 123) <= "1100";

    If falling_edge(clk) AND i>=2 AND Temp(0)/='0' then

      i:= i-2;

      If Temp(i)='0' AND Temp(i-1)='0' then
            For j in 2 to 123 loop
                If Temp(j)='0' then Temp(j-1) <='1';
                                    Temp(j-2) <='0';
                                    Exit;
                End if;
```

```
            End loop;
            X<= "00";
            Y<= "ZZ10";
End if;


If Temp(i)='0' AND Temp(i-1)='1' then
            For j in 3 to 123 loop
                        If Temp(j)='0' then Temp(j-1) <='1';
                                            Temp(j-2) <='0';
                                            Temp(j-3) <='0';
                                            Exit;
                        End if;
            End loop;
            X<= "01";
            Y<= "Z100";
End if;

If Temp(i)='1' AND Temp(i-1)='0' then
            For j in 3 to 123 loop
                        If Temp(j)='0' then Temp(j-1) <='1';
                                            Temp(j-2) <='1';
                                            Temp(j-3) <='0';
                                            Exit;
                        End if;
            End loop;
            X<= "10";
            Y<= "Z110";
End if;

If Temp(i)='1' AND Temp(i-1)='1' then
            For j in 4 to 123 loop
                        If Temp(j)='0' then Temp(j-1) <='1';
                                            Temp(j-2) <='1';
                                            Temp(j-3) <='0';
                                            Temp(j-4) <='0';
                                            Exit;
                        End if;
            End loop;
            X<= "11";
            Y<= "1100";
      End if;

   End if;

      In_K_Unit  <= X;
      Out_K_Unit <= Y;
      K_Seq <= Temp;

   End if;

End Process;
END behavioral;
```

Rabie A. Mahmoud received the B.Sc. Degree, Faculty of Science, Tishreen University, Latakia-Syria, in 2001, the MS. and Ph.D. in Computational Science, Faculty of Science, Cairo University, Egypt, in 2007 and 2011 respectively. Currently, he is with General Organization of Remote Sensing (GORS), Syria and the Department of Computer Engineering, Arab Academy of Science, Technology & Maritime Transport, Latakia branch, Latakia, Syria. His current interests include Cryptography, FPGA Implementations of Cryptography and Data Security Techniques. rabiemah@yahoo.com