FPGA Implementation of Uniform Random Number based on Residue Method

Zulfikar
Electrical Engineering Department, Faculty of Engineering, Syiah Kuala University
Jl. Tgk. Syech Abdul Rauf No. 7, Darussalam, Banda Aceh 23111
e-mail: zulfikarsafrina@yahoo.co.id

***Abstract*—This paper presents the implementation and comparisons of uniform random number on Field Programable Gate Array (FPGA). Uniform random numbers are generated based on residue method. The circuit of generating uniform random number is presented in general view. The circuit is constructed from a multiplexer, a multiplier, buffers and some basic gates. FPGA implementation of the designed circuit has been done into various Xilinx chips. Simulation results are viewed clearly in the paper. Random numbers are generated based on different parameters. Comparisons upon occupied area and maximum frequency from different Xilinx chip are examined. Virtex 7 is the fastest chip and Virtex 4 is the best choice in terms of occupied area. Finally, Uniform random numbers have been generated successfully on FPGA using residue method.**

Keywords: ***FPGA implementation, random number, uniform random number, residue method, Xilinx chips***

***Abstrak*—Paper ini memaparkan implementasi dan perbandingan dari bilangan random uniform pada *Field Programable Gate Array* (FPGA). Bilangan-bilangan random dibangkitkan dengan menggunakan metode *residue*. Rangkaian untuk membangkitkan bilangan random disajikan disini secara umum. Rangkaian tersebut tersusun dari sebuah multiplexser, sebuah multiplier, beberapa buffer dan beberapa gerbang logika dasar. Implementasi pada FPGA dari rangkaian yang telah dirancang telah dilakukan terhadap beberapa variasi chip dari Xilinx. Hasil-hasil simulasi ditampilkan secara mendetail di dalam tulisan ini. Bilangan-bilangan random dibangkitkan dengan menggunakan parameter yang berbeda-beda. Perbandingan terhadap area yang dibutuhkan dan frekuensi maksimum dari berbagai jenis chip Xilinx diteliti. Dalam hal kecepatan, Virtex 7 merupakan pilihan yang terbaik, sementara Virtex 4 lebih optimal dalam hal area yang dibutuhkan. Sehingga dapat disimpulkan bahwa bilangan random uniform telah berhasil dibangkitkan pada FPGA dengan metode *residue*.**

Kata kunci: ***Implementasi FPGA, bilangan random, bilangan random uniform, metode residue, chip-chip Xilinx***

I. INTRODUCTION

Random numbers are used almost in all computer simulations to generate random numbers or samples. The random numbers need to mimic a real condition of any input to computer programs. Uniform random number is the mostly wide used because it is very simple.

Nowadays, many electronic applications do not require a computer or a processor. They have been replaced by an embedded integrated circuit (IC) containing real circuits instead of programs. The applications able to perform tasks faster and more efficient.

Many applications using embedded IC require random numbers. The real circuit for generating random numbers have been proposed [1]-[5].

In this work, we proposed a very simple circuit to generate uniform random numbers. The circuit has been designed based on residue method. The idea has been initialized using Matlab code [6]. The randomness and repetition of numbers generation has been tested using chi-square formulated in [7].

The rest of the paper is organized as follows. Section II deals with theory of how to generate random numbers using residue method. The circuit design is viewed and explained deeply in section III. The next section provides implementations results and a few discussions of this. Conclusions are presented in section IV.

II. BACKGROUND

In general, the probability density function (pdf) of uniform random number variables is given by Eq. (1) [7],[8]:

$$f(x) = \begin{cases} \dfrac{1}{(b-a)}, & a \le x \le b \\ 0, & x < a \text{ and } x > b \end{cases} \quad (1)$$

Uniform distribution of the generated random number is shown in Fig. 1. The numbers between a and b with magnitude 1/(b-a) and the total area of the distribution is 1.

There is a well-known method commonly used to generate random numbers uniformly called residue

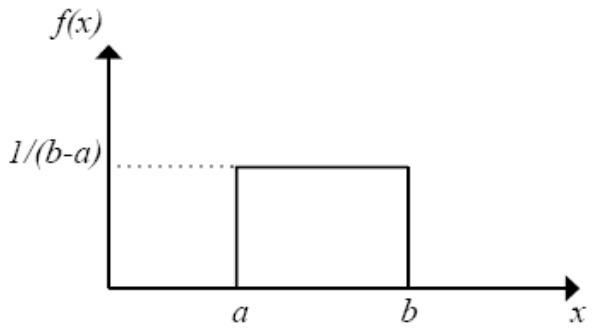

Fig. 1. Uniform random variable distribution

method. The idea of the method is by taking modulo of generated sequence random numbers [9].

$$X_{n+1} = (aX_n + b) \bmod m \quad (2)$$

Parameters *multiplication factor* a, b and m have to be chosen carefully in order to avoid repetition of similar numbers before m. Usually, good results can be obtained by choosing b=0 [9]-[11]. Meanwhile seed (X) should be different every time random numbers are generated.

For instance, with a = 13, c = 0, m = 31, and seed = 1, the sequence begins with 1, 13, 14, 27, 10, 6, 16, 22, 7, 29, 5, 3, ...
The next value looks a bit unpredictable, but it can compute (13 x 3 = 39) mod 31, which is 8 [12].

III. Method

Fig. 2 shows designed circuit to generate uniform random number based on residue method. In the design, it is assumed that b=0. The circuit consist of a multiplexer, a multiplier, two AND gate, a buffer (required clear) and two buffers (required enable). The circuit requires initial value seed given from outside. Input A_in is used to provide a *multiplication factor*. Random numbers are produced serially through port named Output. Port *seed* is used for initial value and only given one at the beginning. Port A_in is used to pass multiplication value into the circuit.

The circuit also equipped with two control signals which are enable and reset. Signal enable control the start operation of the circuit. Initially signal reset have to be HIGH (enable=LOW) to clear all previous stored values in the buffers. Pre-defined value of seed and A_in have to be available at the input ports before enable goes HIGH (reset=LOW). After this, every times clock goes HIGH, a random number produced at the output port.

IV. Results and Discussions

The implementation of the designed circuit is targeted to various Xilinx FPGA chips. Four chips have been selected for this purpose; Spartan 3, Spartan 6, Virtex 4 and Virtex 7.

Figs. 3 and 4 show behavior simulation results of uniform random number generator using 4 bit and 8 bit respectively. Both of the implementations using seed=3 and a=2. The numbers resulted from these implementations

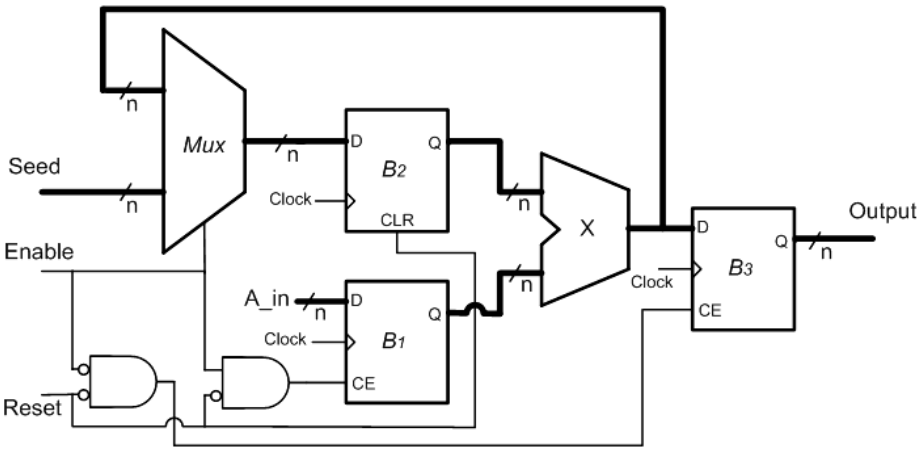

Fig. 2. Design of uniform random number generator based on residue method (b=0)

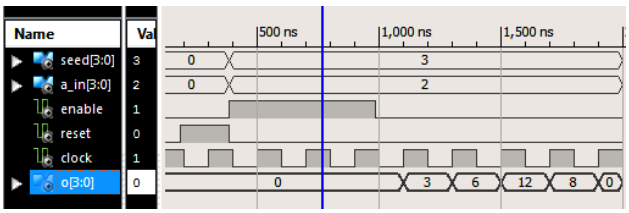

Fig. 3. Behavior simulation of the designed 4 bit uniform random number generator (seed=3, a=2)

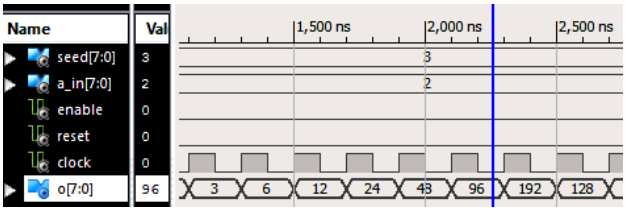

Fig. 4. Behavior simulation of the designed 8 bit uniform random number generator (seed=3, a=2)

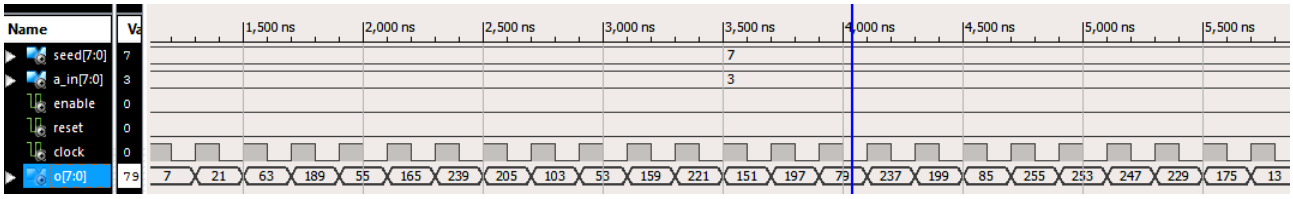

Fig. 5. Behavior simulation of the designed 8-bit uniform random number generator (seed = 7, a = 3)

are not repeated, also they produced few numbers only.

Another simulation (Fig. 5) has been done of 8 bit generator using seed=7 and a=3. It produced more random numbers compare than the previous simulations. Moreover, the numbers are repeated after a very long period. Therefore, the randomness and the amount of number produced are depend on seed and initial *multiplication factor* a.

Table 1 shows speed comparisons in terms of maximum frequency that might be achieved for Spartan 3, Spartan 6, Virtex 4 and Virtex 7. It can be seen that the maximum frequency decreased as the wordlengths increased. The fastest random number produced is when it is implemented into Virtex 7, it would be reached over 700 MHz (*wordlengths* = 4 bit). Conversely, Spartan 3 is the slowest chip of the designed random numbers implementations.

In term of occupied area that is represented by the amount of required slices, it is varied. For *wordlengths* = 4 bit, Spartan 3 chip requires only 6 slices which is around a half of the slices required when the circuit implemented into Spartan 6 and Virtex 7. However, in this case, Virtex 4 is the best chip for random number implementation. All of these can be seen in the Table 2.

Figs. 6 and 7 show other views of speed representation. The graphs provide a more clear figures of speed variation. For implementation into Virtex 7, it can be seen that the speed reduces significantly when wordlengths increased from 4 bit to 8 bit. Changing wordlengths from 8 bit to 16 bit did not have significant impact to the speed.

Table 1. Speed comparisons of various Xilinx chips implemented overrent *wordlengths*

	Maximum Frequency (MHz)				
	4 bit	8 bit	16 bit	24 bit	31 bit
Spartan 3	216	198	172	75	84
Spartan 6	372	209	209	81	81
Virtex 4	381	216	216	120	120
Virtex 7	737	270	270	208	163

Table 2. Occupied area (slice) comparisons of various Xilinx chips implemented over four different wordlengths

	Occupied Area (Slices)				
	4 bit	8 bit	16 bit	24 bit	31 bit
Spartan 3	6	13	26	49	67
Spartan 6	11	16	30	48	62
Virtex 4	9	9	17	28	36
Virtex 7	11	16	30	48	62

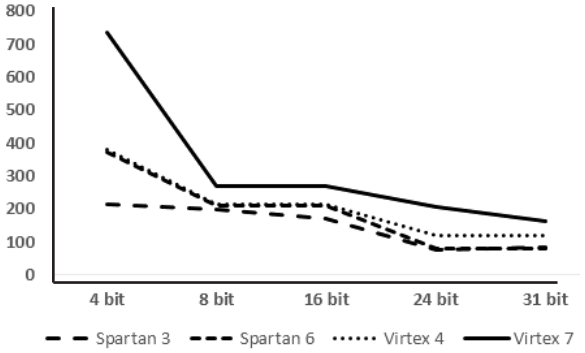

Fig. 6. Speed comparisons of the designed generator for various Xilinx chips

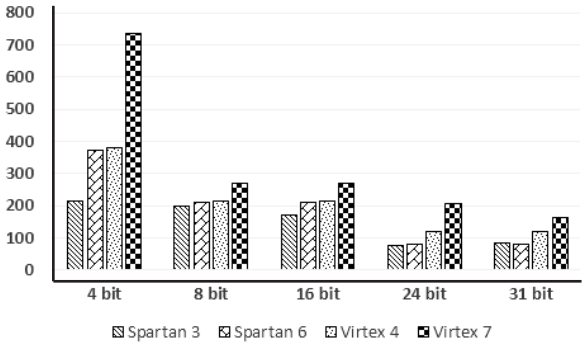

Fig. 7. Speed comparisons of the designed generator implemented using 4 bit, 8 bit, 16 bit, 24 bit, and 31 bit

Figs. 8 and 9 provide information about area occupies in different ways. For Virtex 4 implementation using wordlengths 4 bit and 8 bit, the area required is equal. As stated before, Virtex 4 is the best chip for random number

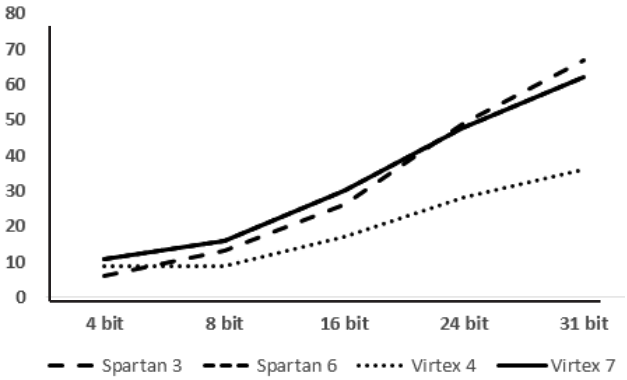

Fig. 8. Area comparisons of the designed generator for various Xilinx chips

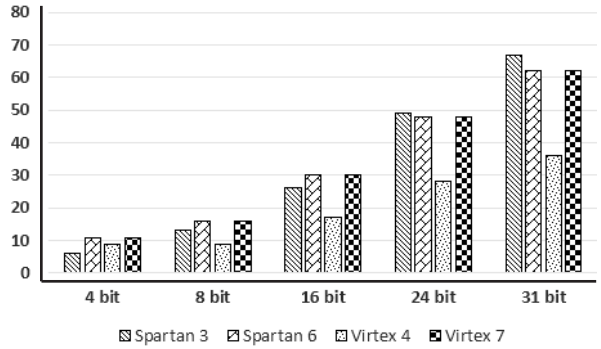

Fig. 9. Area comparisons of the designed generator implemented using 4 bit, 8 bit, 16 bit, 24 bit, and 31 bit

implementation, this figure is shown clearly in the Fig. 9.

Some other implementation data of the designed circuit to Xilinx Virtex 7 using *wordlengths* = 8 about area occupation and maximum frequency are:

```
=====================================
Macro Statistics
# Multipliers                    : 1
 8x8-bit multiplier              : 1
# Registers                      : 24
 Flip-Flops                     : 24
# Multiplexers                   : 1
 8-bit 2-to-1 multiplexer       : 1
=====================================
Slice Logic Utilization:
Number of Slice Registers:       16  out of  437600    0%
Number of Slice LUTs:            10  out of  218800    0%
Number used as Logic:            10  out of  218800    0%

Number of LUT Flip Flop pairs used:  17
Number with an unused Flip Flop:     1  out of    17    5%
Number with an unused LUT:           7  out of    17   41%
Number of fully used LUT-FF pairs:   9  out of    17   52%
Number of unique control sets:       2

IO Utilization:
Number of IOs:                   27
Number of bonded IOBs:           25  out of   300    8%

Specific Feature Utilization:
Number of BUFG/BUFGCTRLs:        1  out of    32    3%
Number of DSP48E1s:              1  out of   840    0%
=========================================

Timing Summary:
---------------
Speed Grade: -2
Minimum period: 3.710ns (Maximum Frequency: 269.513MHz)
Minimum input arrival time before clock: 1.189ns
Maximum output required time after clock: 0.575ns
Maximum combinational path delay: No path found
=====================================
```

It can be seen from the synthesis result into Virtex 4, the designed circuit requires only use some part of logic cells (LCs). The chip has (digital signal processing) DSP, but the designed circuit do not require any DSP at all.

V. Conclusions

FPGA implementation of uniform random number based on residue method has been demonstrated on various Xilinx chips. The circuit for FPGA implementation has been proposed using a multiplexer, a multiplier, two AND gates, and three buffers. The randomness and repetition of produced uniform numbers vary depending on selection values of seed and *multiplication factor* a.

The implementations have been done over four different wordlengths (4 bit, 8 bit, 16 bit and 31 bit). Virtex 7 achieved highest maximum frequency 737 MHz using *wordlengths* = 4 bit. In terms of area occupation, Virtex 4 is the best choice except for *wordlengths* = 4.

References

[1] T. Jennuwein and U. Achleitner, "A fast and compact quantum random number generator," American Institute of Physics, Vol. 71, No. 4, 2000.

[2] K. H. Tsoi, and K. H. Leung, "Compacted FPGA-based and Pseudo Random Generators", Proc. of the 11th Annual IEEE Symposium on Field-Programmable Custom computing Machines (FCCM'03), 2003

[3] V. Fischer and M. Drutarovsky, "True random number generator embedded in reconfigurable hardware", Proceding of CHES 2002, LNCS 2523, pp. 415-430, 2003

[4] H. M. K. Sammy, Y. L. Edmund, "FPGA-based High-speed True Random Number Generator for Cryptographic Applications," Proceding of IEEE TENCON 2006, pp. 1-4, November 2006.

[5] J. Kumar, S. Shukla, D. Prakash, P. Mishra, and S. Kumar, "Random Number Generator Using Various Techniques through VHDL," Transaction of International Journal of Computer Applications in Engineering Sciences, Vol I, Issue II, June 2011.

[6] Zulfikar, "Generating Non Uniform Random Numbers Using Residue and Rejection Methods," Journal Rekayasa Elektrika, Vol. 8 No. 2, October 2009.

[7] A. L. Garcia, "Probability and Random Process for Electrical Engineering," 2nd ed, Addison-Wesley Publishing Company, 1994.

[8] M. Evans, N. Hasting, and B. Peacock, Statistical Distributions, 3rd ed, Wiley series in probability and statistic, 2000.

[9] R. Halprin, M. Naor, "Games for Extracting Randomness," Weizmann Institute of Science. 2009.

[10] G. J. Chaitin, "Exploring Randomness," Springer Publishing, London, 2001.

[11] S. K. Park and K. W. Miller, "Random Number Generators: Good Ones Are Hard to Find," Communication of the ACM, Vol. 31 pp. 1192-1201, 1988.

[12] C. B. Moler, Numerical Computing with MATLAB, SIAM, 2008.