19th International Computer Science and Engineering Conference (ICSEC)
Chiang Mai, Thailand, 23-26 November, 2015

FPGA-based Compact Differential Evolution

Yutana Jewajinda
Department of Electrical Engineering
School of Engineering and Industrial Technology
Silpakorn University
Nakhon Pathom, Thailand
jewajinda_y@su.ac.th

***Abstract*—This paper presents a hardware architecture for compact Differential Evolution (cDE) in FPGA. The proposed architecture includes a novel hardware Gaussian random number generator (GRNG) suitable for real-value compact evolutionary algorithms. The algorithm and hardware architecture of the proposed GRNG are presented. The FPGA-based cDE is designed and implemented. The set of standard benchmark problems are used to test the proposed hardware cDE. The experimental results demonstrate the performance of the proposed hardware architecture in term of speed and FPGA resources.**

Keywords*— *compact differential evolution;hardware; FPGA

I. Introduction

Due to cost and space constraints, solving real-world optimization problems using evolutionary algorithms (EAs) cannot afford full scaled computing power, especially to apply evolutionary optimization to embedded systems in which microcontrollers or FPGA devices are commonly adopted hardware. These embedded systems require real-time capability at lower cost, for example, home automation and robotics or real-time control systems.

Due to limited memory requirement, compact evolutionary algorithms are suitable for embedded systems that have limitation on hardware, space and cost. The compact evolutionary algorithms represent the population of individual solutions using probability distribution, thus reducing requirement on memory hardware.

The compact differential evolution algorithm (cDE) is a real-value compact evolutionary algorithm [1], which is developed from the concept of real-value compact genetic algorithm and a framework of differential evolution [2]. FPGA implementation of traditional DEs and their applications are reported [3,4]. However, there is a limited research literature on hardware cDE.

This paper presents digital hardware architecture of cDE. The proposed architecture includes a GRNG that can also be applied to other real-value compact evolutionary algorithms The cDE algorithm is partitioned into parallelized hardware units. The FPGA-based cDE requires moderate FPGA resources by sharing floating-point hardware units.

II. Background

A. Compact Differential Evolution

The cDE represents the population of solutions using a probability vector (PV) that consists of means, μ and standard deviations, σ. By forming a probability vector (PV) of size n, a PV is an array of Gaussian probability distributions in which each item is characterized by (μ, σ) as follows:

$$PV_t = \{(\mu_1,\sigma_1),(\mu_2,\sigma_2),\ldots,(\mu_n,\sigma_n)\} \tag{1}$$

In order to implement cDE in FPGA, a slightly modified cDE algorithm is presented in Fig.1 [1], in which the square root operation is reduced to multiplication. For updating the standard deviation, the modified part is shown as following equation.

$$\sigma_{i+1} = \sigma_i - \eta(\mu_{t+1} - \mu_t)^2 \tag{2}$$

Where η is used to adjust the rate of decreasing of the standard deviation. Furthermore, for FPGA-based cDE, the Gaussian random number generator (GRNGs) in hardware is necessary and being described in the following section.

B. GRNG hardware

Generalized classification of algorithms for GRNGs can be found in [5]. For digital hardware implementations, there are following techniques: cumulative density function (CDF) inversion [5], transformation using Box-Muller methods [6,7], central limit theorem (CLT) with corrector [8], rejection using ziggurat method [5], recursion using Wallace method [9], and non-piecewise polynomial approximation [5]. From those reported FPGA implementation, the trade-off are between hardware resources, accuracy of Gaussian random numbers, and speed that related to applications of those GRNGs.

Traditionally, digital hardware implementations of GRNGs are aimed for digital communication and recently for Monte-Carlo simulation in hardware [6-11]. To adopt a GRNG method for FPGA implementation of real-value compact evolutionary algorithms, the memory usage is an issue since compact EAs require no memory for population. From those reported FPGA-based GRNGs, there are three methods that require no memory blocks in FPGA, namely CLT-corrector, Table-Hadamard, and piecewise-CLT (PwCLT) [10-12].

```
counter   t = 0
for  I = 1 : n do
   { ** PV  initialization **}
   $\mu[i] = 0$
   $\sigma[i] = \lambda$
end for
generate  elite  by mean of PV
while  not  stopping  criterion  do
      {** Mutation **}
      generate 3 individuals $x_r$, $x_s$, and $x_t$ by mean of PV
      compute  $x'_{off} = x_t + F(x_r - x_s)$
      {** Crossover **}
      $x_{off} = x'_{off}$
      for  i = 1 : n  do
         if rand(0,1) > Cr then
            $x_{off}$[i] = elite[i]
         end if
      end for
      {** Elite selection **}
      [winner, loser] = compete($x_{off}$, elite)
      if $x_{off}$ == winner  then
         elite = $x_{off}$
      end if
      {** PV Update **}
      for i = 1 : n do
         $\mu_{t+1} = \mu_t[i] + \frac{1}{N_D}\left(winner[i] - loser[i]\right)$
         $\sigma_{i+1} = \sigma_i - \eta(\mu_{t+1} - \mu_t)^2$
      end for
      $t = t + 1$
end while
```

Fig. 1. pe-DE/rand/1/bin pseudo-code

The CLT-corrector requires optimization method prior to hardware design for deriving high accuracy corrector [12]. Table-Hadamard also requires tables that contain approximated normal distribution before those tables being corrected to more Guassian distribution using Hadamard transform [10]. PwCLT also requires optimization method to select weights for mixing *n* component distributions [11]. These three methods aimed to produce high accuracy in the tail at least more than 6 sigmas and up to 13 sigmas, aiming for communication and for Monte-Carlo simulation. However, as reported in [13] and [14], the requirement of high quality GRNGs is not necessary for evolutionary algorithm especially for DE. This founding from [13,14] stimulates the author to design an appropriate FPGA-based GRNG for real-value compact evolutionary algorithms. Therefore, we propose a new GRNG for cDE that combined central limit theorem and Hadamard transform called CLT-Hadamard GRNG.

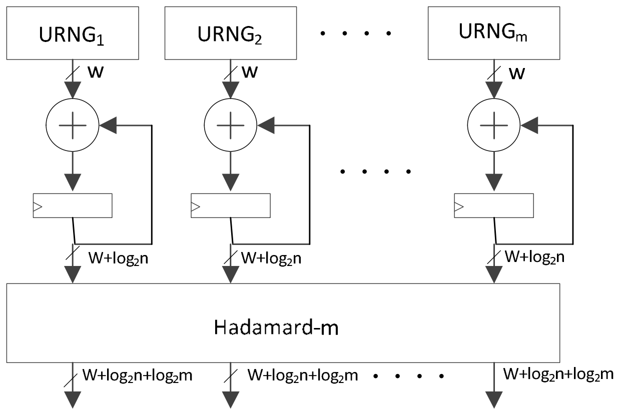

Fig. 2. CLT-Hadamard GRNG

III. CLT-Hadamard GRNG

We propose a parallel GRNG based upon CLT and Hadamard transform (HT) for FPGA-based cDE. The proposed GRNG generates Gaussian random numbers (GRNs) in parallel, as shown in Fig. 2. This parallel generation of GRNs leads to parallel hardware architecture of cDE described later in the fourth section. In addition, the accuracy of GRNs can be adjusted via number of iterations *n* and size of Hadamard matrix: *N*.

From Fig. 2, the first stage of the generator uses CLT as a basis to generate GRNs that are coarsely approximated to the Gaussian. As shown in Fig. 2, the CLT requires *n* additions of uniform random numbers (URNGs). We use LUT-SR as in [15]. In order to generate inputs in parallel for the HT in the second stage, there are parallelized *m* units of URNGs, adders, and accumulators in the first stage. By receiving the GRNs, the HT in the second stage improves the GRNs to more Gaussian.

A. Design and analysis of the first stage: CLT

To analyze the effect of iterative additions of uniform random numbers (GRNs) using CLT, we consider a URNG that generate *n* numbers that uniformly distributed between *–m* and *+m*. Let *y* is the output of the repetitive additions. From [12], the variance and standard deviation are calculated as follows:

$$Var(y) = \sigma_y^2 = \frac{nm^2}{3} \quad (3)$$

$$Std(y) = \sigma_y = \sqrt{\frac{nm^2}{3}} \quad (4)$$

Thus to derive the normal Gaussian distribution, the added output need to be divided by a factor called standard normalization factor (SNF). The SNF is σ_y. The more details of SNF can be found in [12]. Therefore, SNF is used to convert probability density function (pdf) to standard Gaussian distribution.

TABLE I. SNFS FOR DIFFERENT VALUES OF *m* AND *n*

n	m		
	1	2	4
2	0.8164967	1.632993	3.265986
3	1	2	4
4	1.1547005	2.309401	4.618802
8	1.6329932	3.265986	6.531973
12	2	4	8
16	2.3094011	4.618802	9.237604
32	3.2659863	6.531973	13.06395
48	4	8	16

For different values of *m* and *n* as shown in Table I, there are particular cases (n = 3, 12, 48) that SNF are only integer number in multiple of two. This fact leads to replace fixed/floating multiplication by only shift operations for converting GRNs to standard Gaussian.

B. Hadamard Transform

Hadamard Transform (HT) has been extensively used in image and signal processing. However, potential of using HT to generate Gaussian random number are recently reported in [10].

Hadamard transform (HT) of *x(n)* for n = 0,1,2,… N-1 is defined as

$$X(k) = \sum_{n=0}^{N-1} H_N(k,n) \cdot x(n) \qquad (5)$$

Where H_N is Hadamard matrix. Hadamard matrices of higher order are constructed recursively as follows

$$H_N = \begin{bmatrix} H_{N/2} & H_{N/2} \\ H_{N/2} & -H_{N/2} \end{bmatrix} \qquad (6)$$

The Hadamard matrix of order 1 is $H_1 = [1]$ and order 2 is shown as in (7)

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad (7)$$

FPGA implementation of HT has been continuously developed [16,17]. For basic FPGA implementation of HT, the Fast Hadamard Transform (FHT) can be used. The algorithm only requires addition and subtraction. Fig. 3 shows hardware organization of FHT for *N* = 4. From basic block of FHT for *N*=4, the FHT for N =8 can recursively be constructed using FHT with N = 4 as shown in Fig. 4. The latency of the hardware FHT is the depth of addition and subtraction. For N = 4 and 8, the depth is equal to 2 and 3, respectively.

C. Effect of CLT-n and Hadamard Transform to GRNs

Before being corrected by HT, the 65,536 samples of GRNs are generated using CLT with n = 16 as shown in Fig. 5. The 65,536 samples are ranked and put into 128 bins as in [12]. The plotted bar graph is normalized between the highest counted and lowest counted bin and converted from normal distribution to standard Gaussian distribution using the SNF.

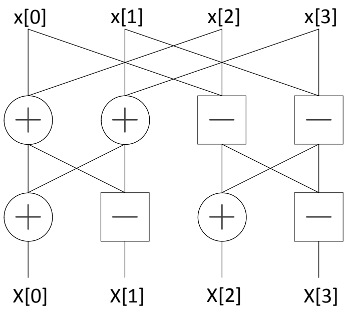

Fig. 3. Hadamard Transform N = 4

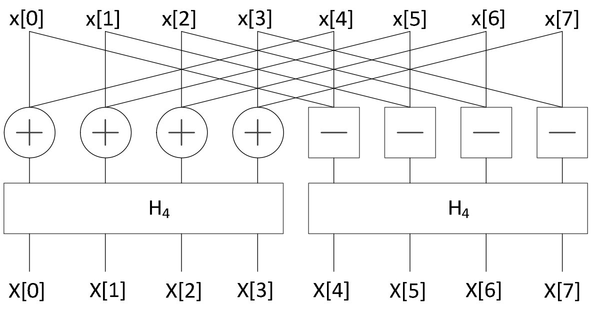

Fig. 4. Hadamard Transform N = 8 constructed from N = 4

From Fig. 5(a) and (b), the standard normal distribution is plotted on the same graph as the histogram of 128 bins in order to be comparable. From Fig. 5(a), the histogram is not perfectly shaped to normal distribution. However, after HT, the more Gaussian is noticeable as in Fig. 5(b). Fig. 6 shows the results of comparison between using Matlab *randn* functions and the proposed method to generate GRNs; by generating the 65,536 samples and using the generated sample to create probability distribution (pdf) using Matlab function (*kdensity*).

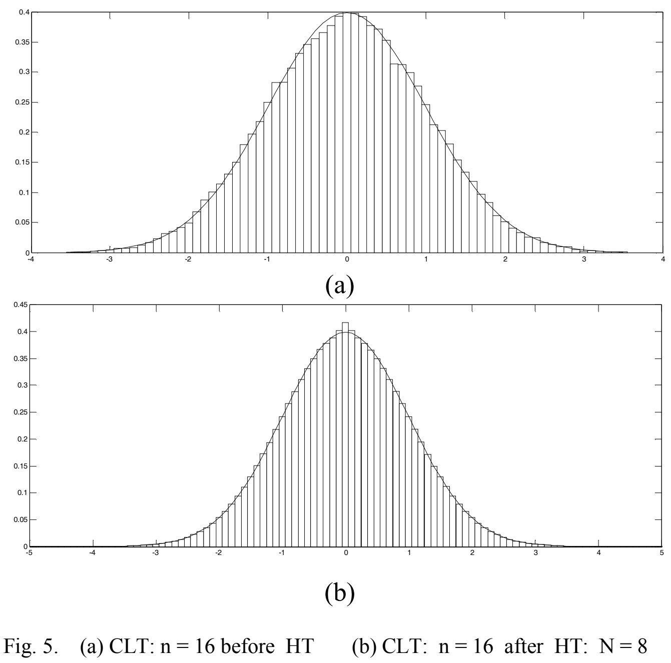

Fig. 5. (a) CLT: n = 16 before HT (b) CLT: n = 16 after HT: N = 8

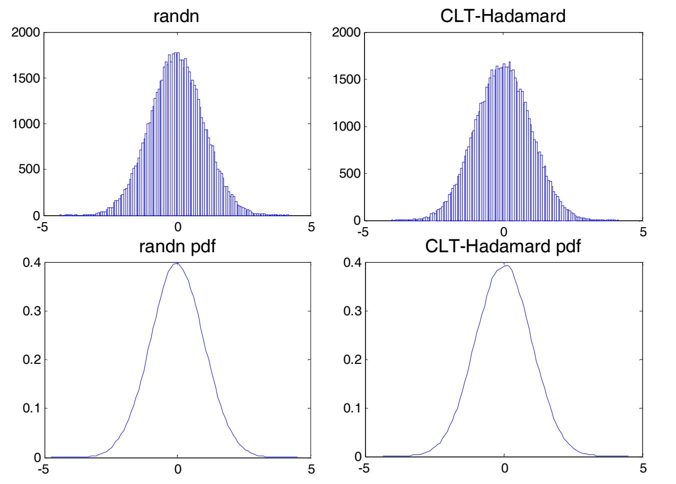

Fig. 6. Comparision between Matlab randn and CLT-Hadamard using 65536 samples and CLT-n = 16 and Hadamard –m = 8

From Fig. 6, the results are closely identical between our proposed GRNG and Matlab *randn.* In order to verify statistical accuracy, we performs chi-square goodness-of-fit test with confidence 0.05 by using fixed sample size of 65,536 and varying number addition *n* of the CLT and size of Hadamard mastrix, *N.* The experiments are performed using Matlab. Table II shows the results of the chi-square test.

The more accurate tail region on both sides after HT can be seen in Fig. 7. From Fig. 7, the tail accuracy is around ±5.5σ, the graph is generated using 1,048,576 samples. Thus, HT increases accuracy at the tail region and improves the bell shape of the generated GRNs.

TABLE II. X^2 TEST RESULTS

CLT n	Hadamard *m*	*p-value*	*Results*
16	8	0.4617±0.2750	Pass
16	16	0.5314±0.3512	Pass
16	32	0.7204±0.2314	Pass
32	8	0.5978±0.2695	Pass
32	16	0.3860±0.3351	Pass
32	32	0.4128±0.4014.	Pass

IV. HARDWARE ARCHITECTURE OF FPGA-BASED CDE

In this section, the digital hardware architecture of cDE is presented.

A. *Top Level Architecture*

Fig. 8 shows the top level architecture of cDE. The proposed architecture consists of six blocks: GRNG, Fixed to Floating Point, Floating Point Datapath, Fitness Evaluation, URNG, and Controller. The architecture employs 32-bit single precision floating point.

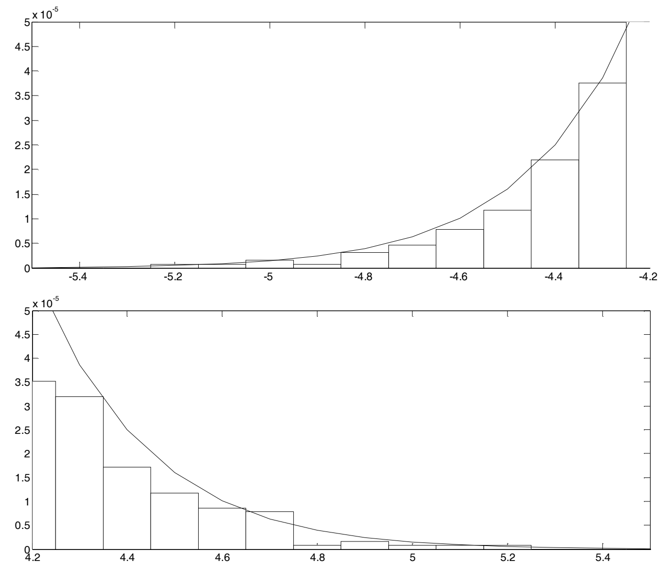

Fig. 7. Tail accuracy between -5.5σ and +5.5, 1,048,576 generated samples

GRNG is implemented using the proposed CLT and Hadamard as described in the third section. The two key parameters are the uniform random numbers to be added together, *n* and Hadamard matrix, *N.* The generated GRNs are in 16-bit format represented in fixed point fractional number valued between 0 and 1. To minimize the FPGA resource, the adder is reused for the CLT stage. Therefore, numbers of adders are equal to *N.*

For fixed point to floating point conversion, Xilinx CoreGen operators are used. The operators convert 16-bit fixed point to 32-bit single precision. In addition, the floating adder/subtractor and floating point multiplier in the floating point datapath have also been generated using Xilinx CoreGen. Since we parallelize the architecture along dimensions, *D,* the number of the floating point datapath required are equal to dimension, *D.*

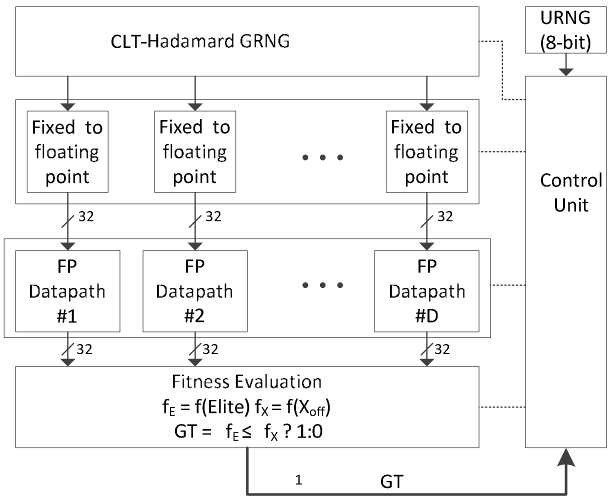

Fig. 8. Hardware Architecture of cDE

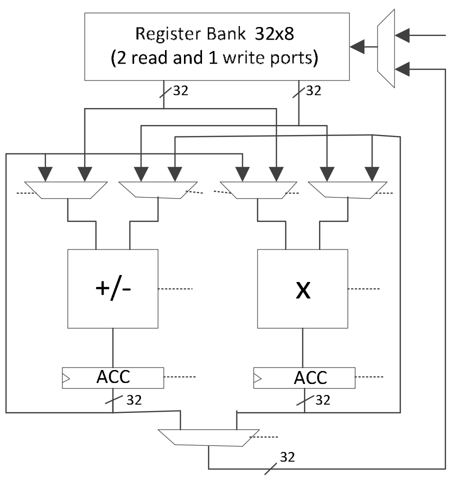

Fig. 9. Hardware Architecture of one Floating Point Datapath

B. *Floating Point Datapath*

The floating point datapath consists of one floating point addition/subtraction unit, one floating point multiplication unit, a 32-bit register bank with two reads and one write ports, two 32-bit registers as accumulators, and multiplexors. All of the 32-bit single precision floating operators are generated using Xilinx CoreGen that requires medium numbers of Xilinx DSP blocks. The floating point datapath operates by control signals from the controller..

C. *Control Unit*

The Control Unit generates control signals to other blocks according to steps in the cDE algorithm. From the cDE algorithm in section two, at the beginning the controller initializes the values in each register bank in all the floating point datapath. These values are $\boldsymbol{x_r}$, $\boldsymbol{x_s}$, $\boldsymbol{x_t}$, *Elite, F*, mean and standard deviation. Then, GRNG generate 3 individuals and the controller operates floating point datapath to generate $\boldsymbol{x_r}$, $\boldsymbol{x_s}$, and $\boldsymbol{x_t}$ by mean and standard deviation stored in the register bank. After that, the mutated vector is computed as $\boldsymbol{x}'_{off} = \boldsymbol{x_t} + F(\boldsymbol{x_r} - \boldsymbol{x_s})$. The URNG as shown in Fig. 8 is used by the controller for crossover operation between $\boldsymbol{x}'_{off}$ and *Elite.* Next, the controller receives the results of the fitness evaluation by the GT bit from the fitness evaluation block. Finally, the controller generates sequence of control signals to find new mean and new standard deviation accordingly.

V. Simulation and Implementation results

There are two experiments. First, the behavioral model of the proposed architecture is tested by selected benchmark problems. Second, the performance of the proposed hardware architecture is compared to software implementation on multi-core processors.

In order to measure performance of the FPGA-based cDE that uses CLT-Hadamard GRNG, five problems from the CEC2005 testbed, see [1,2], are selected as follows:

f_1 Shifted Sphere function
f_2 Shifted Schwefel's Problem 1.2
f_3 Shifted Rotated High Conditioned Eliptic Function
f_4 Shifted Schwefel's Problem 1.2 with Noise
f_5 Schwefels Problem 2.6 with Global Optimum on Bounds

A. *Comparision bettwen the proposed GRNG and a standard GRNGs on the benchmark problems*

Running the benchmark problems, cDE that uses the proposed CLT-Hadamard GRNG are compared to cDE that uses a high-quality GRNG, a ziggurat GRNG [5]. The Q-test described in [14] is applied to experiments shown in Table III. The Q measure is computed as Q = *ne*/*R* where *R* is the percentage of the successful runs, the run is reach a certain predetermined results. And *ne* is the number of fitness evaluations required to reach these thresholds. The 30 independent runs have been performed. For each single run, numbers of fitness evaluations or generations are fixed to maximum at 100,000. The averaged best fitness values are reported in Table III. The results show that the average fitness values between the two cDEs with different GRNG are closely in same digit range.

TABLE III. Average Final Fitness ± Standard deviation for 30D problems

Test Problem	***cDE CLT-n = 16 Hadamard N = 8***	***cDE Ziggurat GRNG***
f_1	6.253e-05±2.35e-05	3.172e-05±1.65e-05
f_2	5.847e-01±4.21e-01	4.529e-01±3.37e-01
f_3	3.051e+06±2.37e+04	1.281e+06±2.39e+04
f_4	9.672 e+02±4.54e+02	5.337 e+02±2.91e+02
f_5	1.052 e+04±5.26e+04	1.813 e+04±3.57e+04

In spite of more detail investigations are needed in term of number of test problems and comprehensively statistical tests; the two methods deliver closely performance. Moreover, the simulation results in this study extend the finding in [13,14] to compact DE that medium GRNG quality can be used for cDE.

B. *Speed up comparision between FPGA-based cDE and software cDE*

In this section, the computation time between FPGA and software implementation on PC computers are measured. The Virtex-5 FPGA running at 300Mhz and software implementation running on Intel Core i-3 and i-7 are compared.

To measure speed up of the FPGA-based cDE, the non-shifted Sphere function is used. The reasons for this is because we only counts clock cycles required in the hardware cDE and not clock cycles used by the fitness evaluation since the test function has to be modeled in behavioral Verilog behavioral SystemVerilog and linked with C function. Also, the fitness evaluation is problem dependent. For the fitness evaluation, the Sphere function is shown as follows:

$$f_1(x) = \sum_{i=1}^{D} z_i^2 \qquad (8)$$

TABLE IV. PERFORMANCE RESULTS FOR THE SPHERE FUNCTION WITH DIMENSION = 16 BY VARYING GENERATIONS

No. of gen.	FPGA 300MHz (ms)	Core-i3 2.2GHz (ms)	Core-i7 3.4GHz (ms)	Speedup over Core-i3	Speedup over Core-i7
1000	0.356	285.174	99.727	801.96	280.45
2000	0.697	552.064	185.029	791.492	265.27
4000	1.413	983.269	301.573	697.578	213.95
8000	2.788	1815.934	541.373	651.408	194.2
10000	3.438	2146.665	651.999	624.345	189.52

TABLE V. GRNG COMPARISON WITH PUBLISHED WORK (ALL ARCHITECTURES USE XILINX VIRTEX-2/4/5 DEVICES)

Design	Logic Slice	Multipiler	Block RAM	Tail Acc.	Speed MSamples/ sec
BM[6]	1528	12	3	8.2σ	466
BM[7]	534	3	2	6.6σ	440
CLT(LTA) [12]	625	7	6	12σ	115
Table-Hadam [10]	102	0	0	8σ	351
Proposed	143	0	0	~5.5σ	483

Table IV shows the speed up of the FPGA-based cDE for problem dimension of 16. The FPGA-based cDE uses CLT: *n* = 16 and Hadamard: N = 8. The experimental results show that the FPGA-based cDE can archive speedup around 600 fold over serial software implementation on Core-i3 and around 200 fold over Core-i7. This increased speedup occurs due to the more numbers of parallel floating point datapath using DSP block inside the FPGA.

Table V shows the FPGA resources used by the proposed GRNG in comparison to the other published works. In spite of the lower tail accuracy, the purposed method requires much lower hardware resources and adequate for cDE algorithm. When compared to Table-Hadamard method, even though Table-Hadamard requires less logic resouces in FPGA, the proposed GRNG is more straight forward, simpler to implement in hardware and not requiring the complicated optimization process as in Table-Hadamard [10] and in [11].

The FPGA resources required for the one dimension of hardware cDE is shown in Table VI. The FPGA resources are moderately utilized. None of the block RAM are required. However, the DSP blocks inside the FPGA are utilized for fast floating point operation

TABLE VI. FPGA RESOURCES FOR ONE DIMENSION (USE XILINX VIRTEX-5 DEVICES)

	Slice LUT	Slice Reg.	DSP 48E1	Blk. RAM	Latency	Freq. Mhz
GRNG	143	68	0	0	20	410
Floating Point Datapath	354	524	4	0	12	372
Main Cntrl.	86	114	0	0	52	410
Total	**583**	**706**	**4**	**0**	**52**	**372**

.

VI. CONCLUSIONS

The FPGA hardware architecture of cDE is presented. The new GRNG is also proposed. This GRNG can also be applied to other real-value compact evolutionary algorithms as well. The presented parallel hardware architecture utilizes the same floating point datapath units in order to minimize the FPGA resources. The experimental results show the performance of the proposed architecture tested with the standard set of benchmark problems. By requiring moderate FPGA resource, this proposed architecture can be further applied for real-value hardware optimization engine using lower cost FPGA devices for FPGA-based embedded systems.

REFERENCES

[1] E. Mininno, F. Neri, F. Cupertino and D. Naso, “Compact Diffential Evolution”, IEEE Trans. Evol., Vol.15, No. 1, pp. 32-53, Feb. 2011.

[2] S. Das and P.N.Suganthan, “Differential Evolution: A Survey of the State-of-the-Art”, IEEE Trans. Evol., Vol.15, No. 1, pp. 1-31, Feb. 2011.

[3] R. Peesapati, KK. Anumandla, S. Kudikala, S. L. Sabat, “SOC based floating point implementation of Differential Evolution using FPGA”, Journal of Design Auto., Vol. 16, no. 4, pp 221-240, 2012.

[4] R. Peesapati, KK. Anumandla, S. Kudikala, S. L. Sabat., “Comparative study of system on chip based solution for floating and fixed point differential evolution algorithm”, Swarm and Evol., No. 19, pp. 68-81, 2014.

[5] D. B. Thomas, W. Luk, P. H. W. Leong and J. D. Villasenor "Gaussian random number generators", ACM Comput. Surv., vol. 39, no. 4, pp.11 2007

[6] D.-U. Lee, J. D. Villasenor, W. Luk and P. H. W. Leong "A hardware Gaussian noise generator using the Box-Muller method and its error analysis", IEEE Trans. Comput., vol. 55, no. 6, pp.659 -671 2006.

[7] A. Alimohammad, S. F. Fard, B. F. Cockburn and C. Schlegel "A compact and accurate Gaussian variate generator", IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 16, no. 5, pp.517 -527 2008

[8] J. S. Malik, J. N. Malik, A. Hemani and N. D. Gohar "Generating high tail accuracy Gaussian random numbers in hardware using central limit theorem", Proc. 19th Int. Conf. VLSI-SoC, pp.60 -65, 2013

[9] D.-U. Lee, W. Luk, J. D. Villasenor, G. Zhang and P. H. W. Leong "A hardware Gaussian noise generator using the Wallace method", IEEE Trans.VLSI Syst., vol. 13, no. 8, pp.911 -920 2005

[10] D.B. Thomas, “Parallel generation of Gaussian random numbers using the table-hadamard transform,” in Proc. FCCM, 2013.

[11] D.B. Thomas, “FPGA Gaussian Random Number Generators with Guaranteed Statistical Accuracy,” in Proc. FCCM, 2014.

[12] JS. Malik, A. Hemani, JN. Malil, B. and NG. Gohar, “Revisiting Central Limit Theorem: Accurate Gaussian Random Number Generation in VLSI”, IEEE Trans. VLSI Sys., Vol. 23, No. 5, May. 2015.

[13] M. Montes, A.V. Rodriguez , “Sensitivity of Evolutionary Algorithms on the Random Number Generator,” ICANNGA 2011, LNCS vol. 6593, pp. 371-380. Springer, 2011

[14] Tirronen, Ville et. al, “Study on the Effects of Pseudorandom Generation Quality on the Performance of Differential Evolution,” ICANNGA 2011, LNCS vol. 6593, pp. 361-370. Springer, Heidelberg, 2011

[15] D. B. Thomas and W. Luk "The LUT-SR family of uniform random number generators for FPGA architectures", IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 21, no. 4, pp.761 -770 2013.

[16] A. Amira, A. Bouridane, P. Milligan, and M. Roula. Novel FPGA implementations of walsh-hadamard transforms for signal processing. IEE Proceedings-Vision, Image, 148(6):377-383, Mar. 2001

[17] Meher, P.K.; Patra, J.C.,” Fully-pipelined efficient architectures for FPGA realization of discrete Hadamard transform”, Proc. Int. Conf Application-Specific, 20