World Academy of Science, Engineering and Technology
International Journal of Electrical and Computer Engineering
Vol:1, No:8, 2007

The Hardware Implementation of a Novel Genetic Algorithm

Zhenhuan Zhu, David Mulvaney, and Vassilios Chouliaras

Abstract**—**This paper presents a novel genetic algorithm, termed the Optimum Individual Monogenetic Algorithm (OIMGA) and describes its hardware implementation. As the monogenetic strategy retains only the optimum individual, the memory requirement is dramatically reduced and no crossover circuitry is needed, thereby ensuring the requisite silicon area is kept to a minimum. Consequently, depending on application requirements, OIMGA allows the investigation of solutions that warrant either larger GA populations or individuals of greater length. The results given in this paper demonstrate that both the performance of OIMGA and its convergence time are superior to those of existing hardware GA implementations. Local convergence is achieved in OIMGA by retaining elite individuals, while population diversity is ensured by continually searching for the best individuals in fresh regions of the search space.

Keywords**—**Genetic algorithms, hardware-based machine learning.

I. Introduction

A number of authors have described GA hardware solutions and applied these to embedded applications, such as sensor data processing [1][2] and algorithm acceleration [3][4][5]. In such implementations, the need to incorporate significant numbers of memory units is recognized as adversely affecting both execution speed and the physical silicon area required. For example, in the roulette GA [6], memory is required to store the population data and the fitness values. Such memory could be provided on-chip, in which case it is likely to operate at full clock speed, but occupy significant physical area that could otherwise have been used for other GAs or processing elements. The alternative is to provide off-chip memory, in which case not only may cost considerations dictate the use of slower memory requiring a number of clock cycles to access, but also, if a number of GAs are combined in a single device, it is unlikely that the data bandwidth will be sufficient to allow all GAs simultaneous access to their respective populations. The compact GA [3] is one approach specifically designed to address this memory bottleneck issue. To permit their use in a wide range of applications, implementations of GAs need to be flexible in their structure to allow variations in the population size and the lengths of individuals [3][4]. For example, a suitable length for the individuals is typically influenced by the size of the solution space and the diversity of the population is often related to the number of individuals in that population. One important measure of the performance of GAs is their rate of convergence. A hardware implementation based on 'half-siblings-and-a-clone' [1] was shown to shorten the GA convergence time.

The OIMGA algorithm introduced in this paper is specifically designed to address the issues of memory requirement and memory access speed. In addition to achieving this, the results also demonstrate that OIMGA is flexible in its structure, maintains diversity in its population and its rate of convergence is significantly faster than that of the existing GA hardware methods described above.

Authors are with Department of Electronic and Electrical Engineering, Loughborough University, Loughborough, LE11 3TU, UK (e-mail: {z.zhu2, d.j.mulvaney, v.a.chouliaras}@lboro.ac.uk).

II. OIMGA Algorithm

OIMGA incorporates two searches that interact hierarchically, namely a global search and a local search. In the global search, regions are selected sequentially from the entire search space for more detailed exploration by the local search. In the global search, a single individual is maintained (termed **topChrom**) that is the best (according to the fitness criterion) obtained from all the local searches carried out so far. The local search investigates the regions selected by the global search in order to determine the local optimum individual (LOI). This is achieved by generating an initial population in a narrow range using micro mutation. If the micro mutation results in a better individual this becomes the new LOI. The process is repeated until a termination criterion is satisfied.

As the best individual among all generations that have been investigated is always kept, then the proof given by Radolph [7] can be applied directly to demonstrate that OIMGA is convergent. As the algorithm repeatedly initializes the population space following a global search, OIMGA is very effective in maintaining diversity and preventing premature convergence.

Compared with the existing methods described above, the convergence time of OIMGA is likely to be shorter due to reductions both in the total search space explored and in the population size [8]. A further execution speed enhancement in the hardware implementation is also easily identifiable since the executions of the global and the local searches are prime candidates for hardware pipelining. Table 1 shows the parameters available to a designer using the OIMGA algorithm, while Fig. 1 shows the pseudocode of OIMGA itself.

TABLE I
OIMGA PARAMETERS

l	individual length
n	population size
m	the size of the miniature space around the LOI
t_gens	maximum number of consecutive global generations without improvement
k_gens	maximum number of consecutive local generations without improvement
d_adjustor	range of mutation of an individual
pm	probability of mutation

```
g=t_gens;
while g>0 % start a global search
  k=k_gens;
  d=d_adjustor;
  for i=1:n  % find an individual with best fitness
    loiChrom=[rand(1,l)<0.5]; % random l-bit individual
    loiFit=fitness(loiChrom,l);  % find its fitness
    if loiFit>bestFit     % keep an elite individual and
      bestChrom=loiChrom; % its fitness
      bestFit=loiFit;
    end
  end
  while k>0 % start a local search
    exchange=0; % number of exchanges of tempChrom and
                % bestChrom
    for i=1:m   % perform local search m times
      tempChrom=bestChrom;
      for j=d:l  % produce a micro mutation in the
                 % range d to l
        if rand<pm    % 'rand' is a random number
          tempChrom(j)=not(tempChrom(j));
                      % invert the jth bit
        end
      end
      tempFit=fitness(tempChrom,l);
      if tempFit>bestFit  % keep elite local individual
        bestChrom=tempChrom; % individual and
        bestFit=tempFit;    % its fitness
        exchange=exchange+1;
        k=k_gens;   % restore k value
        d=d_adjustor;   % restore d value
      end
    end
    if exchange=0 % decrease range (d-l) if no exchange
      d=d+1;
    end
    k=k-1;
  end
  if bestFit>topFit   % global evaluation - keep elite
    topChrom=bestChrom;  % individual and its fitness
    topFit=bestFit;
    g=t_gens;
  end
  g=g-1;
end
```

Fig. 1 The pseudocode for the OIMGA algorithm

III. IMGA HARDWARE DESIGN

Fig. 2 shows the main structure of the hardware implementation of OIMGA. The **LOI-generator** initiates the local process by randomly producing a population that includes *n* individuals, and then searches for the LOI. In the **micro-mutation unit,** the individuals are allowed to evolve in value only within the range indicated by the value of **d_adjustor** and any change of range is controlled by the **d_controler.** The fitness value of the generated individual is calculated by the **fitness-unit** and the **local-evaluator** compares the fitness of the current LOI with that of the previous one and replaces it if its fitness is better. The search in the local space is repeated *m* times. If, during these searches, a new LOI is not found then the range that **d_adjustor** indicates is decreased. Should the fitness of the LOI not improve over *k_gens* cycles, then the LOI is sent to the **global evaluator**. The global evaluator implements the global process and retains the globally best individual and its fitness value found from all the local searches. The global process terminates when the fitness has not improved over *t_gens* operations of the local process.

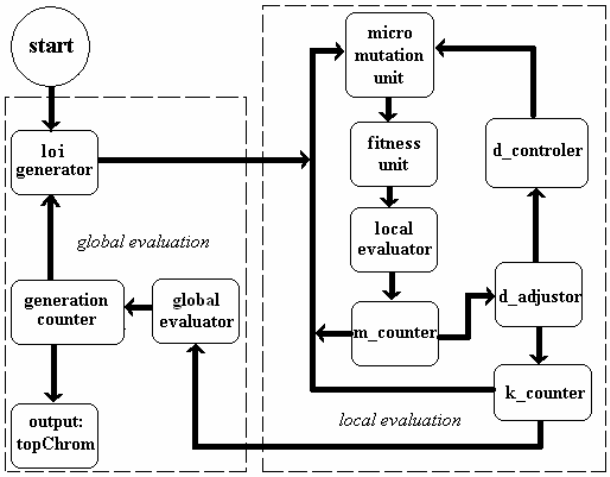


Fig. 2 The main structure of OIMGA

A. LOI Generator

The LOI generator shown in Fig. 3 includes a random number generator **RNG** that produces an *l*-bit random individual whose fitness value is calculated by the **fitness unit** and stored in the register **loiFit**. The unit **cmp1** is used to compare the fitness of **loiFit** with that of the best fitness value held in **bestFit** and, if it is better, **bestFit** is replaced by **loiFit** and the new individual (of length *l*) replaces that held in the register **bestChrom**. The **n bit counter** ensures that the entire process is carried out *n* times, where *n* is the population size. Note that in order to modify the size of the population, it is only necessary to change the length of the counter.

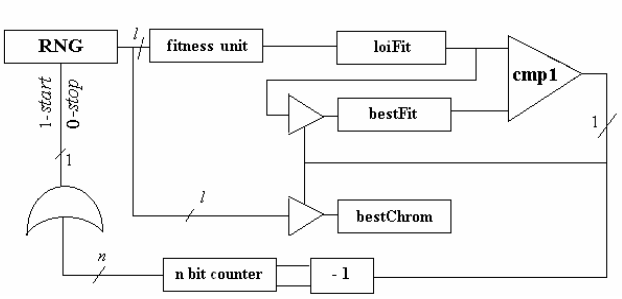


Fig. 3 LOI generator

B. Micro-mutation Unit

The micro-mutation unit is shown in Fig. 4. If the probability of mutation **pm** is greater than **RNG***i* and **d_MRSR***i* is set, the i^{th} bit of **bestChrom** is mutated. The register **tempChrom** holds the value of the chromosome following mutation and is evaluated in the **fitness unit**. If its fitness is better than that in **bestFit** (as determined in the **local evaluator** shown in Fig. 5), the signal **cmp2** operates the tri-state gate to replace **bestChrom** by the value in **tempChrom**. To modify the length of the individual, a corresponding change can be made to the number of bits in the micro mutation unit.

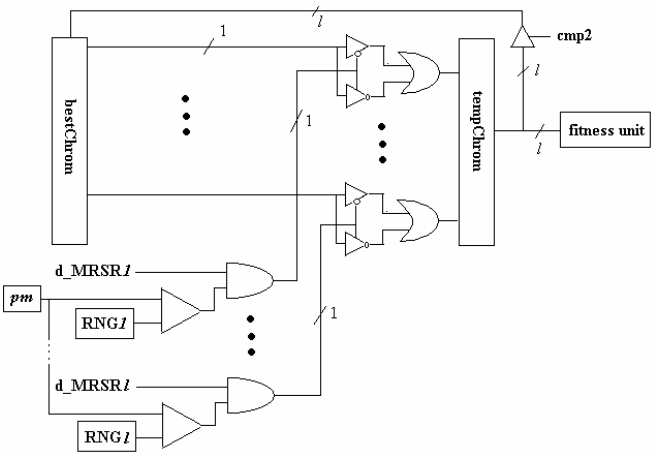


Fig. 4 Micro mutation unit

C. Local Evaluator

The local evaluator shown in Fig. 5 uses the fitness values to select the better individual from **tempFit** and **bestFit**, and keeps this elite individual and its fitness value during local evolution.

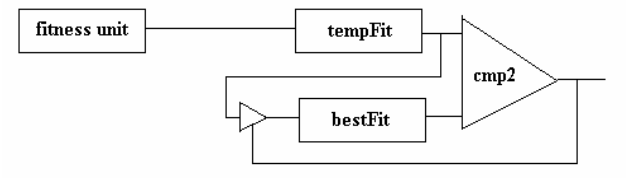


Fig. 5 Local evaluator

D. Adjusting the Range of Mutation

During an evolution process, generally the times between modifications to the fitness values decrease, indicating that the evolution is converging to a final value. To speed up convergence, it is appropriate to reduce the allowed change of mutation values in order to investigate the space in the more immediate vicinity of the current best individual.

Initially, the bits in the **mask right shift register** shown in Fig. 6 are all set, $MRSR_i=1$, $i\in[1,l]$. The initial value of the range held in **d_initial** is set to a predefined value, signifying that all but this number of bits in the individuals should be mutated. This value is copied into **d_counter**. The exchange register (**exchange**) is initialized with 0 and is incremented whenever the local evaluator replaces the current best individual. The value in **d_counter** defines the number of shifts that are performed by the MRSR (with the left-most bit zero filled); at each shift **d_counter** decrements by 1. To understand the operation, consider the case where the initial value held in **d_counter** is 3. In this case, following the shift operations, the state of MRSR is shown as follows.

$$d\_MRSR_i = \begin{cases} 0 & 0 \le i \le 3 \\ 1 & 4 \le i \le l \end{cases} \quad (1)$$

These values indicate that the range of mutation is in [4, *l*]. During local evolution based on LOI, **exchange** will be increased by 1 if **bestChrom** and **bestFit** are replaced. After each generation of local evolution, **d_initial** will increase by 1 if **exchange** is still 0, thereby reducing the number of bits that are mutated in the micromutation unit.

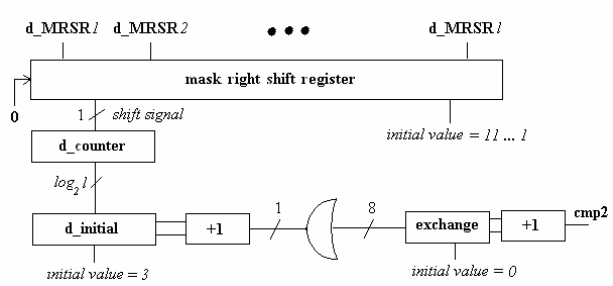


Fig. 6 Circuitry to adjust the range of mutation

E. Global Evaluator

The principle of operation of the **global evaluator**, shown in Fig. 7, is very similar to that of the local evaluator. The global evaluator selects the better individual from **bestFit** and **topFit**, and keeps the elite individual from all generations and its corresponding fitness value in **topChrom** and **topFit** respectively.

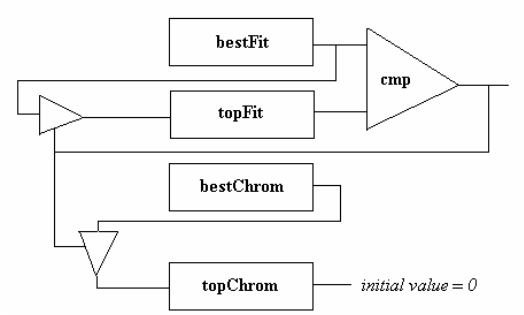


Fig. 7 Global evaluator

IV. Results

To evaluate the efficiency of a number of hardware implementations of GAs with OIMGA, namely half-siblings-and-a-clone [1], roulette [6] and compact GA [3], the two benchmark functions defined by Zhang and Zhang [9] shown in the following equations were used.

$$f_1(x)=|x(1-x)^2 sin(200\pi x)| \qquad x\in(0,1) \qquad (2)$$

$$f_2(x)=(1-2\sin^{20}(3\pi x)+\sin^{20}(20\pi x))^{20} \qquad x\in(0,1) \qquad (3)$$

$f_1(x)$ has 200 local maximum and minimum values in its defined range (Fig. 8), while $f_2(x)$ has 20 local maximum and minimum values in its defined range (Fig. 9). It is very difficult to determine analytically the maximum and minimum values of the two functions by methods other than using some form of search [9].

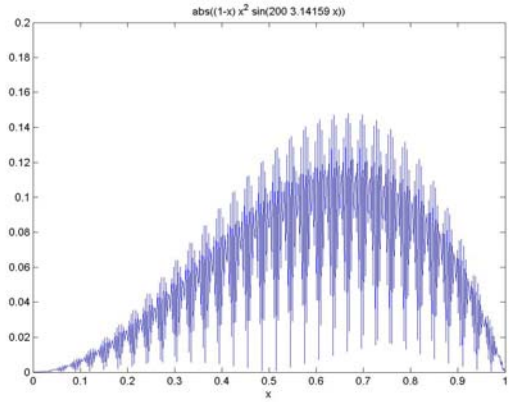


Fig. 8 Benchmark function $f_1(x)$

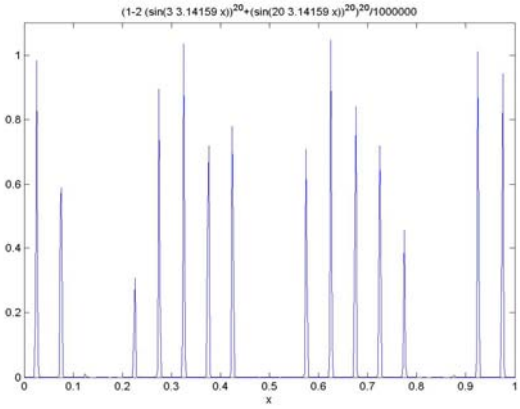
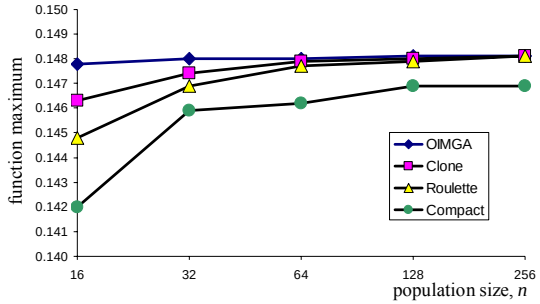


Fig. 9 Benchmark function $f_2(x)$

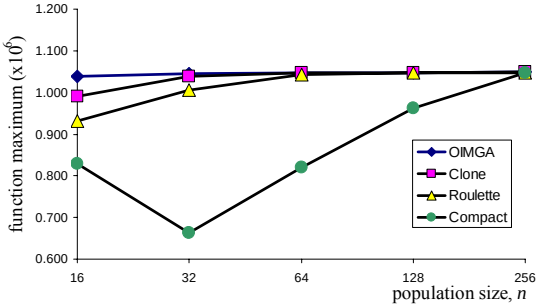
Presented here are results of a number of experiments to assess the following three aspects of the four hardware GA implementations: the quality of the solution produced, the calculation time and the hardware component requirements.

The GA implementations (other than OIMGA) were carried out according to the descriptions given by the respective authors. The simulations were all developed and run in MATLAB [10] on the same host computer system. Since MATLAB cannot fully reproduce the cycle-accurate timings of a hardware implementation, the timings can only be regarded as indicative.

In the first set of experiments, the performance of the GAs in determining the maximum values of the functions $f_1(x)$ and $f_2(x)$ were investigated for various values of population size and individual lengths. Fig. 10 shows that for a fixed individual length, OIMGA outperformed the other GA implementations, particularly for small populations. The performance of the compact GA was noticeably inferior to the other implementations. The poor performance of the compact GA was also apparent when the population size was fixed and the maximum function values determined for a range of lengths of the individuals, Fig. 11. The remaining three GAs all performed similarly under this test.

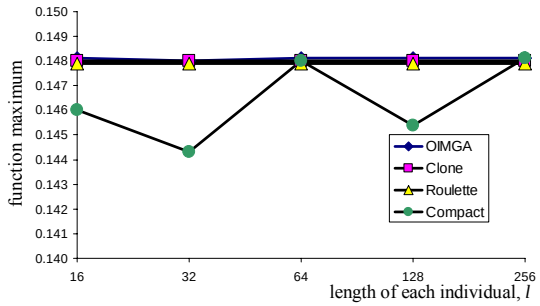


(a) Estimated $f_1(x)$ maxima

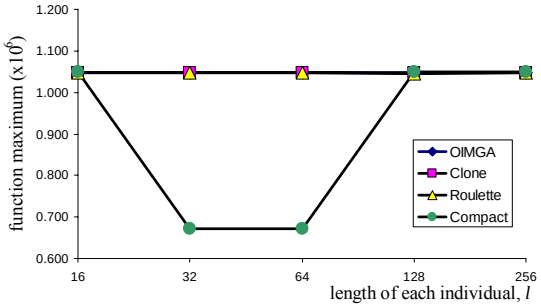


(b) Estimated $f_2(x)$ maxima

Fig. 10 Maxima of the benchmark functions found by the GAs for a range of population sizes and at a fixed individual length (l) of 32. Each data point shown was calculated from results averaged over 200 tests, except for the compact GA where only 20 tests were carried out



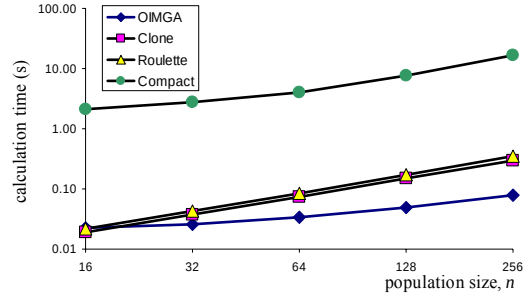
(a) Estimated $f_1(x)$ maxima



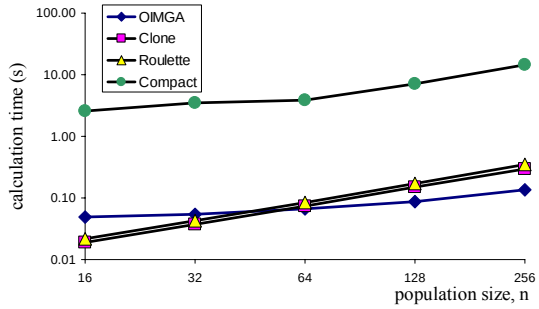
(b) Estimated $f_2(x)$ maxima

Fig. 11 Maxima of the benchmark functions found by the GAs for a range of individual lengths and at a fixed population size (n) of 128. Each data point shown was calculated from results averaged over 200 tests, except for the compact GA where only one test was carried out

The second set of experiments investigated the calculation times to reach convergence when determining the maximum values of the functions $f_1(x)$ and $f_2(x)$ for the different population sizes and individual lengths. In Fig. 12, it can be seen that the compact GA performed poorly across a range of population sizes, with the calculation times often being two orders of magnitude greater than those of the other GAs. It can be seen from Fig. 12 that, as the population size is increased, the calculation times of OIMGA increase less steeply than those of the other GAs. More detailed investigations revealed that, with the doubling of the population size, the calculation times for OIMGA increased at only half the rate of the half-siblings-and-a-clone and the roulette GAs. Fig. 13 shows that the calculation times for the compact GA were particularly long when the length of the individuals was increased beyond 32. These results also show that the other GA methods produced shorter calculation times and OIMGA performed particularly well in the more demanding cases where the individuals were of greater length and the population larger.



(a) Estimated $f_1(x)$ calculation times

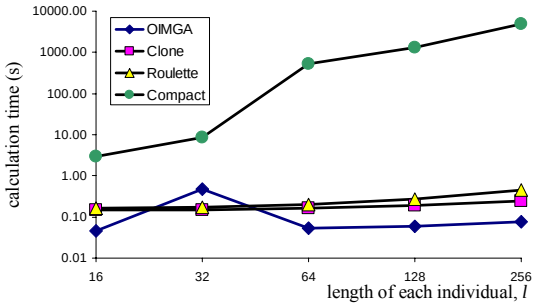


(b) Estimated $f_2(x)$ calculation times

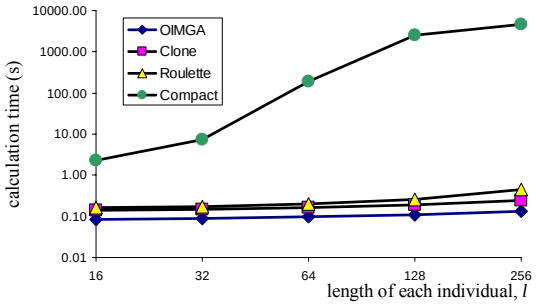
Fig. 12 Calculation times of the benchmark functions found by the GAs for a range of population sizes and at a fixed individual length (l) of 32. Each data point shown was calculated from results averaged over 200 tests, except for the compact GA where only 20 tests were carried out

In order to generate representative figures, the experimental procedure to produce the results involved adjustment of the respective parameters of each of the GAs (other than for l and m whose values were purposely varied to obtain the results). The parameters used by OIMGA for the estimation of the maximum values of $f_1(x)$ and $f_2(x)$ are given in Table II. Note that altering the width of the fitness value affects not only the system performance, but also has an effect on other hardware requirements, such as the width of the comparator.

Hardware implementations of GAs mainly consist of random number generators, comparators, registers and memory. The requirement of each component can be described with its total bit number (TBN). For example, if there are ten 8-bit registers in a circuit, their TBN is 80 bits. To illustrate the relative complexities of the GAs investigated in the current work, the values in Table 3 were obtained from algorithmic estimates of the hardware requirements of four different classes of component. It can be seen that the TBN for the compact GA and OIMGA solutions are an order of magnitude less than those for the other GA methods. However, in contrast with OIMGA, the modest hardware requirement of the compact GA has clearly been achieved at the expense of performance.



(a) Estimated $f_1(x)$ calculation times



(b) Estimated $f_2(x)$ calculation times

Fig. 13 Calculation times of the benchmark functions found by the GAs for a range of individual lengths and at a fixed population size (n) of 128. Each data point shown was calculated from results averaged over 200 tests, except for the compact GA where only one test was carried out

TABLE II
OIMGA PARAMETER VALUES

Function	m	t_gens	k_gens	$d\_adjuster$	pm	width of fitness value
$f_1(x)$	10	4	5	3	0.382	32
$f_2(x)$	16	6	5	4	0.382	32

TABLE III
HARDWARE REQUIREMENTS OF THE GA IMPLEMENTATIONS

GA	random number generators	comparators	registers	memory	total
OIMGA	160	224	296	0	680
Clone	32	96	256	4096	4480
Roulette	59	64	478	8192	8793
Compact	256	262	352	0	870

V. CONCLUSION

The paper has introduced a new GA algorithm that is particularly suited for hardware implementation because of its minimal memory requirement and its ability to allow both the size of the population and the length of the individuals to be altered simply by replicating existing logic units. When run on benchmark problems, the new algorithm compared favorably with other hardware solutions found in the literature, both in terms of its execution time and in its performance on benchmark problems. Future publications will present the results of our investigations of implementing the GAs in a hardware design language and running cycle-accurate simulations in order to determine more precisely their relative performances.

REFERENCES

[1] Sharawi, M.S., Quinlan, J. and Abdel-Aty-Zohdy, H.S., "A hardware implementation of genetic algorithms for measurement characterization", IEEE 9th International Conference of Electronics, Circuits, and Systems, Dubrovnik, Croatia, 3, 2002, pp.1267-1270.

[2] Hauser, J.W. and Purdy, C.N., "Sensor data processing using genetic algorithms", IEEE Mid- West Symp. on Circuits and Systems, August 2000.

[3] Aporntewan, C. and Chongstitvatana, P., "A hardware implementation of the compact genetic algorithm", 2001 IEEE Congress on Evolutionary Computation, Seoul, Korea, 2001, pp.27-30.

[4] Wakabayashi, S., Koide, T., Toshine, N., Yamane, M. and Ueno, H., "Genetic algorithm accelerator GAA-II", *Proc.* Asia and South Pacific Design Automation Conference, Yokohama, Japan, January 2000.

[5] Scott, S.D., Samal, A. and Seth, S., "HGA: A hardware-based genetic algorithm", *Proc.* 3rd ACM/SIGDA Int. Symp. on FPGAs, 1995, pp.53-59.

[6] Ramamurthy, P. and Vasanth, J., "VLSI implementation of genetic algorithms" (under review).

[7] Radolph, G., "Convergence analysis of canonical genetic algorithms", *IEEE Trans. Neural Networks*, 5(1), 1994, pp.96-101.

[8] Li, J. and Wang, S., "Optimum family genetic algorithm", *Journal of Xi'an Jiao Tong University*, 38, Jan 2004.

[9] Zhang, L. and Zhang, B., "Research on the mechanism of genetic algorithms", *Journal of Software*, 11(7), 2000.

[10] Matlab, http://www.mathworks.com/.