EVOLVABLE HARDWARE DESIGN OF SEQUENTIAL LOGIC CIRCUIT

Wesam M.Ahmed1, Abubaker Kashada2
2 ،1Surman College of Science & Technology, Libya
1wesam_ahmed@scst.edu.ly
2kashada@scst.edu.ly

Abstract

Evolvable Hardware (EHW) refers to the generation of electronic circuits using evolutionary algorithms (in our case genetic algorithms).Genetic algorithm (GA) is a robust search algorithm loosely based on population genetics. It effectively seeks solutions from a vast search space at reasonable computation costs. Before a GA starts, a set of candidate solutions, represented as binary bit strings, are prepared. This set is referred to as a population, and each candidate solution within the set as a chromosome. A fitness function is also defined which represents the problem to be solved in terms of criteria to be optimized. The chromosomes then undergo a process of evaluation, selection, and reproduction. In the evaluation stage, the chromosomes are tested according to the fitness function. The results of this evaluation are then used to weight the random selection of chromosome in favor of the fitter ones for the final stage of reproduction. In this final stage, new generations of the chromosomes are "evolved" through genetic operations which attempt to pass on better characteristics to the next generation. Through this process, which can be repeated as many times as required, less fit chromosomes are gradually expelled from a population and the fitter chromosomes become more likely to emerge as the final solution.

Keywords:*Evolvable Hardware; Genetic algorithm;Evolutionary algorithms;sequential logic circuit.*

1.Introduction

The modeling methodology presented in this paper has been adopted for the early steps of the whole development process through which the evolvable chip is obtained. Also this high level of simulation and modeling makes possible prediction that, the evolvable hardware solution is possible to the given task, i.e. image compression [1]. Hardware Description languages (HDL) is selected toward hardware implementation because it is a rich standard language and many well supported tools for efficient simulation and modeling are exist [2].

The use of HDL language in simulation and modeling of systems allows designing parameterized designs, possibility of module reuse, easy management of large designs, enhanced readability, and designs written independently of the technologies used for its final realization.

In a sequential logic circuit, the outputs depend not just on the current values of the inputs, but also on past values of the inputs. The circuit has memory. Sequential circuits can do two things that combinational circuits cannot: they can recognize sequences of inputs and they can generate sequences of outputs. Efforts have been done to evolve the sequential logic circuits [3]

A sequential logic circuit can be made by adding feedback to a combinational logic circuit:

2. Genetic Algorithms

The automated synthesis of digital logic to satisfy the function specification is a well-researched area [4, 5, 6]. This section presents the main operation and features of a conventional GA.

To apply evolution to the design of circuits, one or more candidate designs are described by a string of bits within a memory pool. This memory keeps the so called phenotype or population, which initially consists of randomly selected binary-strings named chromosomes, individuals or genotype and may encode the configuration of reprogrammable arithmetic and logic units or reprogrammable interconnections between hardware components. Each one of these chromosomes is evaluated in terms of how accurately the given configuration (chromosome) approaches the targeted functionality. The evaluation mechanism is called the fitness-function and incorporates one or more objectives that characterize the ideal behavior of a circuit

[7]. Moreover, GAs employ simple operators during reproduction, such as crossover and mutation. At each reproduction two new chromosomes are produced (offspring's) from the parents, which are selected based on their fitness-score from the current population. The aim of the GA is to potentially produce fitter chromosomes compared with those comprising the old population. This iterative process is repeated until an acceptable solution is found or a specified number of iterations, called generations, have been completed. Figure 2 depicts the design flow of a generic GA. According to this, the first population is initialized with random candidate designs.

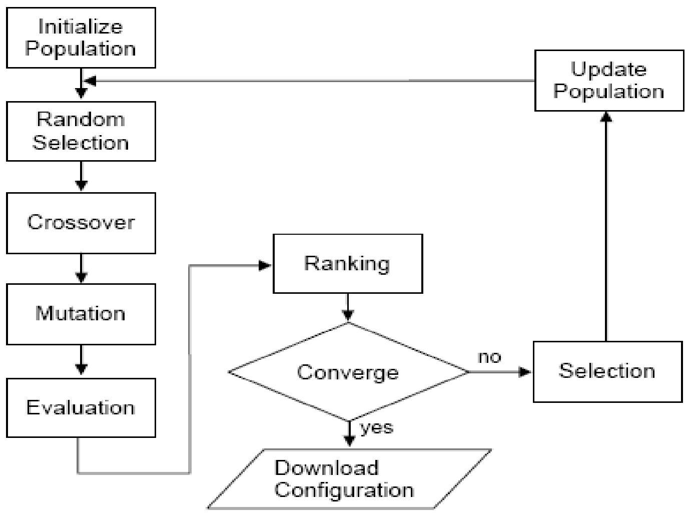

Figure 1:Design flow of a generic GA

The Algorithms

1. Randomly generate an initial population M(0)
2. Compute and save the fitness u(m) for each individual m in the current population M(t)
3. Define selection probabilities p(m) for each individual m in M(t) so that p(m) is proportional to u(m)
4. Generate M(t+1) by probabilistically selecting individuals from M(t) to produce offspring via genetic operators
5. Repeat step 2 until satisfying solution is obtained.

3. Evolvable Hardware

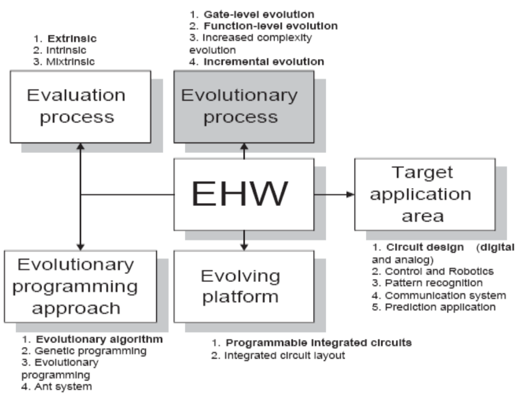
1. Extrinsic
2. Intrinsic
3. Mixtrinsic
1. Gate-level evolution
2. Function-level evolution
3. Increased complexity evolution
4. Incremental evolution
Evaluation process
Evolutionary process
EHW
Target application area
Evolutionary programming approach
Evolving platform
1. Circuit design (digital and analog)
2. Control and Robotics
3. Pattern recognition
4. Communication system
5. Prediction application
1. Evolutionary algorithm
2. Genetic programming
3. Evolutionary programming
4. Ant system
1. Programmable integrated circuits
2. Integrated circuit layout

Figure 2: Taxonomy of evolvable hardware

Evolvable Hardware (EHW) techniques apply machine-learning methods to the automated design, configuration, or repair of electronic devices [8]. While EHW is a new and growing field, it has been successfully applied to obtain useful results in a variety of digital logic and arithmetic circuit applications as well as amplifier circuits, antenna designs and other areas. In this laboratory assignment.

Here, the fundamental concept behind EHW is to use Genetic Algorithms to obtain FPGA configurations autonomously. GAs utilize principles gleaned from natural evolutionary processes as an optimization method where the fundamental premise is that the better-suited offspring from the current generation will be chosen to be present in the succeeding generation, leading to a gradual but steady increase in the capabilities of the individuals in the population [9].

Population-based GAs for EHW start out with a random or semi-random population of designs, each of which constitute potential design candidates that solve the problem at hand. These designs are then optimized by applying genetic operators. Genetic operators include crossover of useful traits from 2 existing designs and random mutation that adds new traits to individuals in the population. Ideally, after applying these genetic operators and selecting the best circuits that result from them, each solution is more optimal than its predecessors.

4. Evolutionary Algorithms

Evolutionary Algorithm theory and show the corresponding between Evolutionary Algorithm and classic design model. Evolutionary Algorithms has appeared as a general concept for developing new computational models for optimization and design [10]. The principles of evolution in nature have been modeled in a variety of different ways and thus a number of computational models have been developed. These are referred to as Evolutionary Algorithms. Evolutionary algorithms have a common conceptual basis associated with the simulation of two fundamental processes [11]. These are the processes of random variation and selection within a population, described by Charles Darwin (1859) as the principles of evolution.

The interplay of variation and selection gradually pulls the population to a target that in evolutionary computation is merely the solution of a computational problem. Thus evolutionary search is employed to solve difficult problems in optimization and design. In many problems, evolutionary algorithms have been found to produce solutions that are better than those produced by the traditional design and other search techniques. Solutions obtained by evolution are often unusual in construction, since they are generated in a completely different manner from the conventional methods for optimization and design. This concept of evolutionary design of efficient and novel solutions has also been adopted in the electronic circuit. Design (see Figure 3).

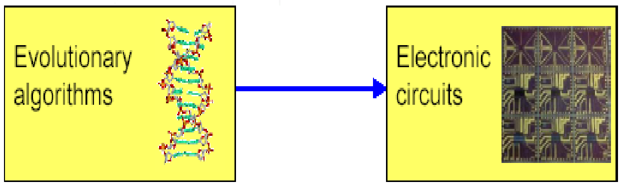

Figure 3. Use of the evolutionaryalgorithms (EA) to create electronic circuits

5.Genetic Operators

Five phases are considered in a genetic algorithm.

Initial Population

The process begins with a set of individuals which is called a Population. Each individual is a solution to the problem you want to solve.An individual is characterized by a set of parameters (variables) known as Genes. Genes are joined into a string to form a Chromosome (solution). In a genetic algorithm, the set of genes of an individual is represented using a string, in terms of an alphabet. Usually, binary values are used (string of 1s and 0s). We say that we encode the genes in a chromosome.

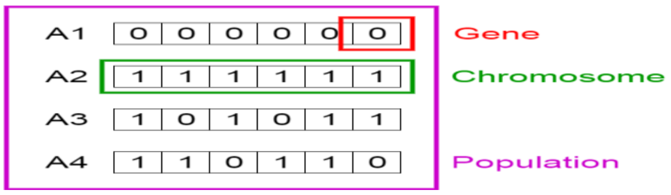

Figure 4.Population, Chromosomes and Genes

2. Fitness Function

The fitness function determines how fit an individual is (the ability of an individual to compete with other individuals). It gives a fitness score to each individual. The probability that an individual will be selected for reproduction is based on its fitness score.

3. Selection

The idea of selection phase is to select the fittest individuals and let them pass their Two pairs of individuals (parents) are selected based on .genes to the next generation their fitness scores. Individuals with high fitness have more chance to be selected for reproduction.

4. Crossover

Crossover is the most significant phase in a genetic algorithm. For each pair of parents to be mated, a crossover point is chosen at random from within the genes. For example, consider the crossover point to be 3 as shown below

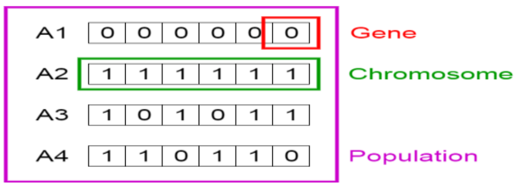

Figure 5.Crossover point

Offspring are created by exchanging the genes of parents among themselves until the crossover point is reached.

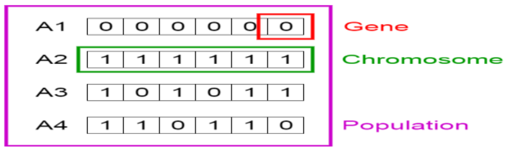

Figure 6. Exchanging genes among parents

The new offspring are added to the population

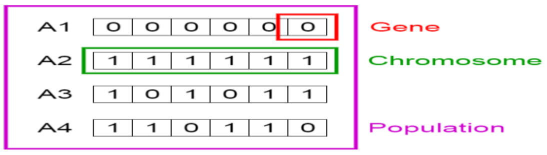

Figure 6. New offspring

5.Mutation

In certain new offspring formed, some of their genes can be subjected to a mutation with a low random probability. This implies that some of the bits in the bit string can be flipped.

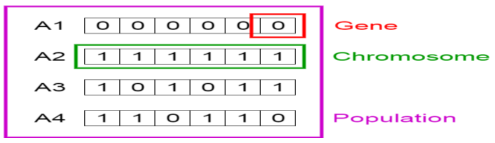

Figure 7. Mutation: Before and After

Mutation occurs to maintain diversity within the population and prevent premature convergence.

1.Benchmark Set

A subset of the PLA benchmark set given in Table 2 is used in this evaluation. The table shows the benchmarks used, together with their number of input/ output/products.

Benchmark	# Input	# Output	# p
Add1c.pla	**2**	**2**	**8**
Add2_3c.pla	**3**	**2**	**8**
Mult2.pla	**4**	**4**	**16**
Exam1.pla	**2**	**1**	**8**
Exam2.pla	**4**	**2**	**8**
Exam3.pla	**4**	**2**	**8**
Exam4.pla	**4**	**2**	**8**

Table 2. Statistics of benchmark examples

2. Crossover and mutation

Crossover and mutation probabilities are dealt with as a single unit, as they are similar and each has a significant effect on a good choice of the other.

Benchmark Examples	P_c	P_m	No. Gates
add1.pla	**0.20**	**0.07**	**6**
	0.10	**0.01**	**5**
	0.30	**0.05**	**4**
	0.50	**0.05**	**6**
	0.80	**0.05**	**6**
add2_3c.pla	**0.20**	**0.07**	**7**
	0.10	**0.01**	**4**
	0.30	**0.05**	**5**
	0.50	**0.05**	**6**
	0.80	**0.05**	**6**
mult2.pla	**0.20**	**0.07**	**7**
	0.10	**0.01**	**6**
	0.30	**0.05**	**7**
	0.50	**0.05**	**6**
	0.80	**0.05**	**7**

Table 3. Compression crossover and mutation.

Note: (P_m) is mutation rate and (Pc) probabilities of crossover rate.

Table 3 shows the effect of P_c and P_m on the number of gates. Figure 7 shows the plot of different probability of crossover P_c with the number of gates. The plot of probability of mutation P_m and the number of gates is shown in figure 8. However if

P_c or P_m are too low, few changes are made from generation to generation, leading to static population.

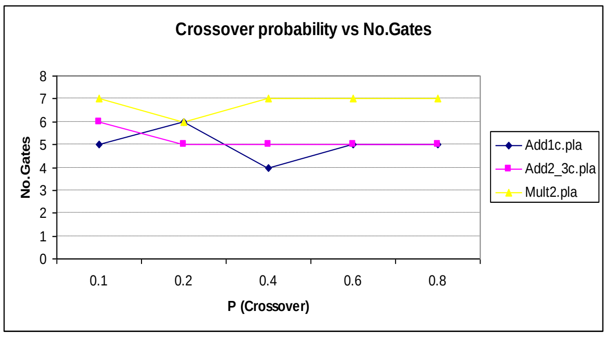

Figure 7. Effect of P (Crossover)

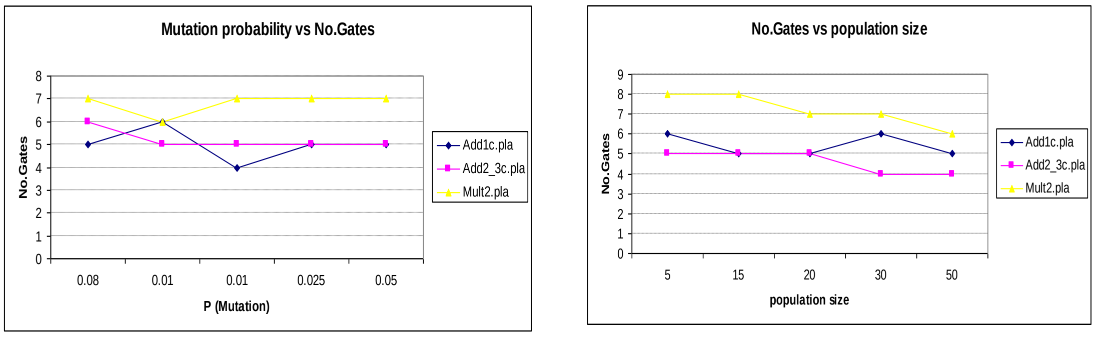

Figure 8. Effect of P (Mutation)

3. Population Size

The probability of crossover and mutation were then fixed at the values given above, and attention was turned to discovering a good value for population size. Increasing in the population size increase its diversity and reduces the probability that the GA will prematurely converge to a local optimum, but it also increases the time required for the population to converge to the optimal regions in the search space. The template is designed so that author affiliations are not repeated each time for multiple authors of the same affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization). This template was designed for two affiliations.

The population size was varied from an initial setting (5) by doubling and halving to explore the value around 15. As shown in table 3.

The aggregation of results has been treated in the same way as in the previous section. Also plotted are the numbers of gates (No. Gates) Vs Population sizes in figure 9. The plots shows that increasing the population size will, on average, lead to decreased number of gates.

Benchmark Example	Population_ size	No. Gates
add1.pla	**5**	**5**
	10	**5**
	20	**4**
	30	**6**
	40	**5**
Add2_3c.pla	**5**	**4**
	10	**5**
	20	**4**
	30	**4**
	40	**4**
mult2.pla	**5**	**6**
	10	**7**
	20	**5**
	30	**7**
	40	**6**

Table 3. Population size numbers of gates and number of literal

4. GAs Generation

In this section, we compare the effect of the number of generations on three benchmarks with the crossover and mutation rate fixed. The results obtained are tabulated and plotted as shown in figure 10. In the table 3, number of generations was varied from 10 to 500. It was clear this range is large enough to see the effect of this parameter on the GA.

Benchmark examples	N0. Generation	No.Gates
add1.pla	10	5
	20	6
	50	4
	100	4
	500	4
Add2_3c.pla	10	5
	20	6
	50	5
	100	4
	500	5
mult2.pla	10	7
	20	8
	50	6
	100	6
	500	5

Table 4. Result number of generation.

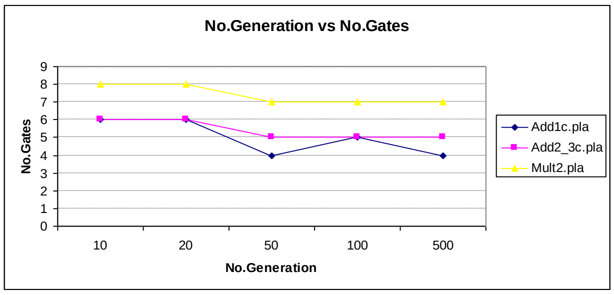

Figure 9. Effect of Generation Size.

From tabulated and graphical results it was decided to choose the number of generation =100. This gave us better results in less time.

6.Comparison of results

The second set of experiments aims to compare the GA results with human design .These comparisons were based on the number of the gates.

The genetic algorithm with population size of 20 was demonstrated for the probability of crossover is 0.25 and the probability of mutation is 0.01. The number of generations over which GA was run was 100. The number of generations required for GA to operate effectively was lower somewhere between 50 and 100. The number of generations was over 100 and it took less than 10 minutes of CPU time to run. The results obtained from the GAs are compared with human design in terms of two inputs gates required in Table 5.

Benchmarks Examples	Number of product	Genetic algorithm	Human design
Add1c	8	4 Gates	6 Gates
Mult2	16	8 Gates	13 Gates
Exam1	8	4 Gates	6 Gates
Exam2	16	8 Gates	12 Gates
Exam3	6	4 Gates	7 Gates
Exam4	16	8 Gates	12 Gates

Table 5.Comparison of Gas with Human designer based on the number of gates.

The main observation to make from these tables is that, the genetic algorithm produces superior solutions to the logic circuit problem compared with human designer. In all benchmark tested in Table 5. the GA produce better results compared to human designer Experimental results For the 7 benchmark tested showed that the GA could generate logic circuit, which required on average 15.44% fewer gates. The area is estimated as the sum of gates area.

7.Conclusion

EHW proposed as a new method for designing circuits for complex real world applications.

One of the problems has been that only small and simple circuits have been evolvable. This paper highlights some of the reasons that can explain why EHW has not yet been widely applied. Further, to make EHW more application, an increased complexity scheme is proposed, where evolving smaller sub-circuits evolves a circuit. Extrinsic evolvable hardware still has two useful features not provided by intrinsic evolvable hardware. Extrinsic evolvable hardware can work with any design paradigm, and so any bias towards a particular kind of circuit behavior is limited only by the researcher's imagination and the quality of the simulation used for evaluation.

8. References

[1] D. Goldbery, “Genetic Algorithm in Search, Optimization, and Machine Learning,” Addison Wesley,1989.

[2] Almaini A.E.A. and Zhuang N., and Bourset F. Minimization of multioutput binary decision diagrams using hybrid generic algorithm. IEEE Electronic Letters, 31 (20): 1722-1723, 1995.

[3] H. Liang, W. Luo and X. Wang, “A tree-step decomposition method for the evolutionary design of sequential logic circuits,” in Genet Program Evolvable Mach, vol.10,pp.231-262, 2009.

[4] P. K. Lala, Practical Digital Design and Testing, Prentice Hall, 1996.

[5] S. Louis, ‘‘Genetic algorithm as computational tool for design,’’ PhD Dissertation, Department of Computer Science, Indiana University, 1993.

[6] A. Thompson, ‘‘An evolved circuit, intrinsic in silicon, entwined with physics,’’ in Proceedings of the 1st International Conference on Evolvable Systems: From Biology to Hardware (ICES96), Lecture Notes in Computer Science, T. Higuchi, et al. (eds.) Springer-Verlag, vol. 1259, 1997, pp. 390–405.

[7] Darringer, j., et. Al., (1984) LSS: A system for production of logic synthesis. IBM J. Res. Develop. Vol. 28, pp. 537-545.

[8] Almaini A.E.A. and Zhuang N. Using genetic algorithms for the variable ordering of reed-muller binary decision diagrams. Microelectronic Journal, 26 (4): 471-480, 1995.

[9] Bystrov A. and Almaini A.E.A. Testability and test compaction for decision diagram circuits. IEEE Proceedings on Circuits, Devices and Systems., 146 (4): 153-158, 1999.

[10] Miller J.F., Kalganova T., Lipnitskaya N., and Job D. The genetic algorithm as a discovery engine: Strange circuits and new principles. In Proc. of the AISB,99 Symposium on Creative Evolutionary Systems, CES,99, ISBN 1-902956-03-6, page 65-74. Edinburgh, UK, The Society for the Study of Artificial Intelligence and Simulation of Behaviour, April 1999.

[11] Murakawa M., Yoshizawa S., Kajitani I., Furuya T., Iwata M., and Higuchi T. Hardware evolution at function level. In Proc. of the Fifth International Conference on Parallel Problem Solving from Nature (PPSNIV), Lecture Notes in Computer Scienec. Springer-Verlag. Heidelberg, 1996.