IJDACR
ISSN: 2319-4863

IJDACR
International Journal Of Digital Application & Contemporary Research

International Journal of Digital Application & Contemporary research
Website: www.ijdacr.com (Volume 1, Issue 7, February 2013)

A VHDL Implementation of Genetic Processor for Random Process Optimization

Sachin Rai
sachinraisagar@gmail.com

Awadhesh K. G. Kandu
awadheshkandu@yahoo.co.in

Abstract – **Genetic algorithm is the most widely used genetic algorithm. This paper presents a VHDL implementation of the genetic algorithm. In results we shown successful implementation of GA and shown its application to calculate the square root of a number.**

Keywords – **Genetic Algorithm, VHDL, Square root.**

I. Introduction

Genetic Algorithms (GAs) was invented by John Holland. Holland proposed GA as a heuristic method based on "Survival of the fittest". GA was discovered as a useful tool for search and optimization problems.

Most often one is looking for the best solution in a specific set of solutions. The space of all feasible solutions (the set of solutions among which the desired solution resides) is called search space (also state space). Each and every point in the search space represents one possible solution. Therefore each possible solution can be "marked" by its fitness value, depending on the problem definition.

With Genetic Algorithm one looks for the best solution among a number of possible solutions represented by one point in the search space i.e.; GAs are used to search the search space for the best solution e.g., minimum. The difficulties in this ease are the local minima and the starting point of the search.

Significance of Genetic Algorithm

Genetic Algorithm raises a couple of important features. First it is a stochastic algorithm; randomness as an essential role in genetic algorithms. Both selection and reproduction needs random procedures. A second very important point is that genetic algorithms always consider a population of solutions. Keeping in memory more than a single solution at each iteration offers a lot of advantages. The algorithm can recombine different solutions to get better ones and so, it can use the benefits of assortment. A population base algorithm is also very amenable for parallelization. The robustness of the algorithm should also be mentioned as something essential for the algorithm success. Robustness refers to the ability to perform consistently well on a broad range of problem types. There is no particular requirement on the problem before using GAs, so it can be applied to resolve any problem. All those features make GA a really powerful optimization tool.

II. Genetic Algorithm

GA handles a population of possible solutions. Each solution is represented through a chromosome, which is just an abstract representation. Coding all the possible solutions into a chromosome is the first part, but certainly not the most straightforward one of a Genetic Algorithm. A set of reproduction operators has to be determined, too. Reproduction operators are applied directly on the chromosomes, and are used to perform mutations and recombination over solutions of the problem.

Appropriate representation and reproduction operators are really something determinant, as the behaviour of the GA is extremely dependents on it. Frequently, it can be extremely difficult to find a representation, which respects the structure of the search space and reproduction operators, which are coherent and relevant according to the properties of the problems.

Selection is supposed to be able to compare each individual in the population. Selection is done by using a fitness function. Each chromosome has an associated value corresponding to the fitness of the solution it represents. The fitness should correspond to an evaluation of how good the candidate solution is. The optimal solution is the one, which maximizes the fitness function. Genetic Algorithms deal with the problems that maximize the fitness function. But, if the problem consists in minimizing a cost function, the adaptation is quite

IJDACR
ISSN: 2319-4863

IJDACR
International Journal Of Digital Application & Contemporary Research

International Journal of Digital Application & Contemporary research
Website: www.ijdacr.com (Volume 1, Issue 7, February 2013)

easy. Either the cost function can be transformed into a fitness function, for example by inverting it; or the selection can be adapted in such way that they consider individuals with low evaluation functions as better.

Once the reproduction and the fitness function have been properly defined, a Genetic Algorithm is evolved according to the same basic structure. It starts by generating an initial population of chromosomes. This first population must offer a wide diversity of genetic materials. The gene pool should be as large as possible so that any solution of the search space can be engendered. Generally, the initial population is generated randomly.

Then, the genetic algorithm loops over an iteration process to make the population evolve. Each iteration consists of the following steps:

Evaluation: Initially many individual solutions are randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Traditionally, the population is generated randomly, allowing the entire range of possible solutions. Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

Truncation Selection: Truncation selection is a selection method used in genetic algorithms to select potential candidate solutions for recombination. In truncation selection the candidate solutions are ordered by fitness, and some proportion of the fittest individuals are selected and reproduced 1/p times.

Crossover: Crossover is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next.

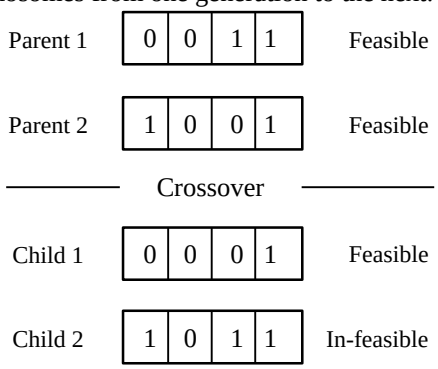

Figure 1: Crossover Operation

Assume a problem of four items has a full feasible random population. When it performs crossover using two feasible solution as parents, it generates to children, it could happen that one of it or both are not feasible as shown in figure 1.

It is analogous to reproduction and biological crossover, upon which genetic algorithms are based. Cross over is a process of taking more than one parent solutions and producing a child solution from them.

Mutation: Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. It is analogous to biological mutation.

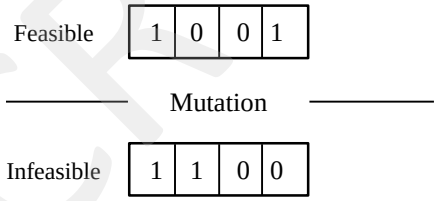

Figure 2: Mutation Operation

Mutation alters one or more gene values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution. Hence GA can come to better solution by using mutation.

The basic genetic algorithm is as follows:

• [start] Genetic random population of n chromosomes (suitable solutions for the problem)

• [Fitness] Evaluate the fitness f(x) of each chromosome x in the population

• New population] Create a new population by repeating following steps until the New population is complete

- [selection] select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to get selected).

- [crossover] with a crossover probability, cross over the parents to form new offspring (children). If no crossover was performed, offspring is the exact copy of parents.

- [Mutation] with a mutation probability, mutate new offspring at each locus (position in chromosome)

- [Accepting] Place new offspring in the new population.

• [Replace] Use new generated population for a further sum of the algorithm.

IJDACR
ISSN: 2319-4863

IJDACR
International Journal Of Digital Application & Contemporary Research

International Journal of Digital Application & Contemporary research
Website: www.ijdacr.com (Volume 1, Issue 7, February 2013)

• [Test] if the end condition is satisfied, stop, and return the best solution in current population.
• [Loop] Go to step2 for fitness evaluation.

The Genetic algorithm process is discussed through the GA cycle

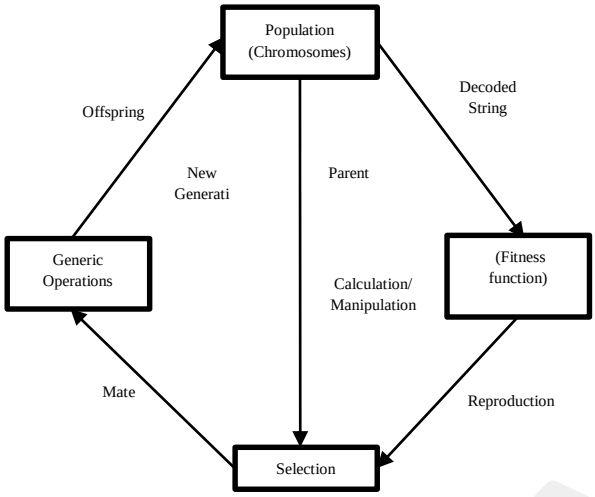

Figure 3: Genetic Algorithm cycle

Reproduction is the process by which the genetic material in two or more parent is combined to obtain one or more offspring. In fitness evaluation step, the individual's quality is assessed. Mutation is performed to one individual to produce a new version of it where some of the original genetic material has been randomly changed. Selection process helps to decide which individuals are to be used for reproduction and mutation in order to produce new search points.

The flowchart showing the process of GA is as shown in Figure 4. Before implementing GAs it is important to understand few guidelines for designing a general search algorithm i.e. a global optimization algorithm based on the properties of the fitness landscape and the most common optimization method types:

1. Determinism: A purely deterministic search may have an extremely high variance in solution quality because it may soon get stuck in worst case situations from which it is incapable to escape because of its determinism. This can be avoided, but it is a well-known fact that the observation of the worst-case situation is not guaranteed to be possible in general.

2. Non-determinism: A stochastic search method usually does not suffer from the above potential worst case, "wolf trap" phenomenon. It is therefore likely that a search method should be stochastic, but it may well contain a substantial portion of determinism, however. In principle it is enough to have as much non-determinism as to be able to avoid the worst-case wolf traps.

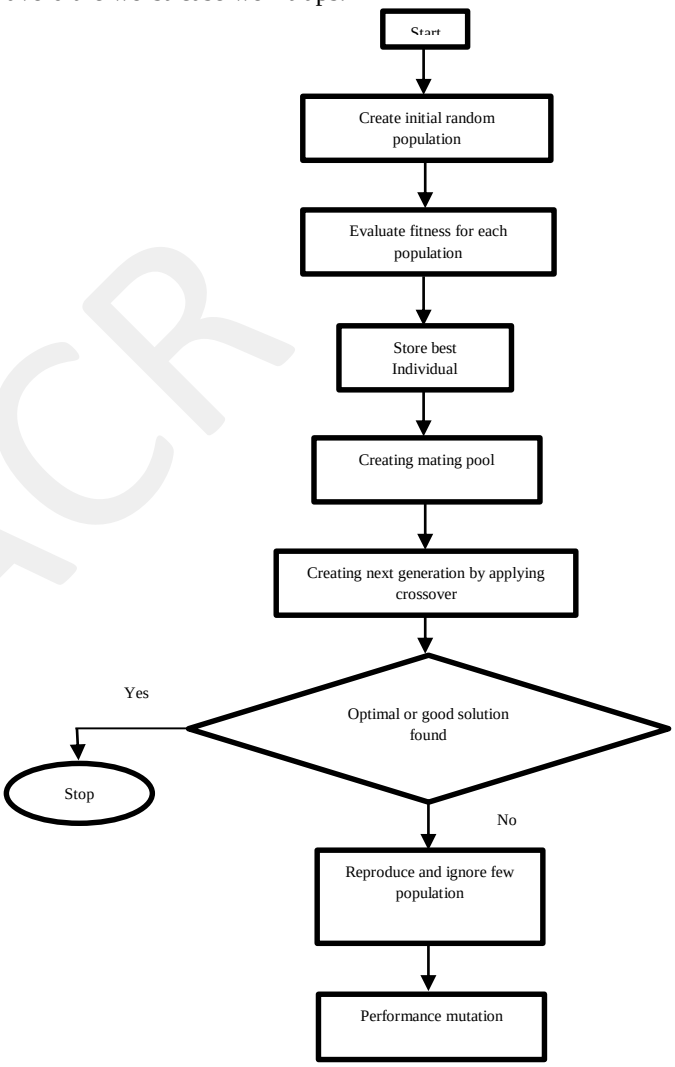

Figure 4: Flow chart of GA

3. Local determinism: A purely stochastic method is usually quite slow. It is therefore reasonable to do as much as possible efficient deterministic predictions of the most promising directions of (local) proceedings. This is called local hill climbing or greedy search according to the obvious strategies.

Based on the foregoing discussion, the important criteria for GA approach can be formulated as given below:

IJDACR
ISSN: 2319-4863

IJDACR
International Journal Of Digital Application & Contemporary Research

International Journal of Digital Application & Contemporary research
Website: www.ijdacr.com (Volume 1, Issue 7, February 2013)

- Completeness: Any solution should have its encoding
- Non redundancy: Codes and solutions should correspond one to one
- Soundness: Any code (produced by genetic operators) should have its corresponding solution
- Characteristic perseverance: Offspring should inherit useful characteristics from parents.

In short, the basic four steps used in simple Genetic Algorithm to solve a problem are,

1. The representation of the problem
2. The fitness calculation
3. Various variables and parameters involved in controlling the algorithm
4. The representation of result and the way of terminating the algorithm.

III. ALGORITHM

The block diagram of proposed work is shown in figure below:

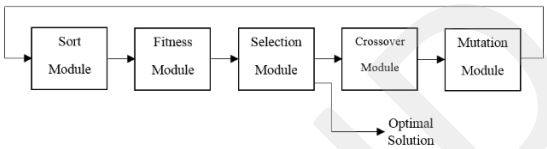

Figure 5: Block diagram of proposed Genetic Algorithm

The processor consists of the following modules:

- Sort Module
- Fitness Module
- Selection Module
- Crossover Module
- Mutation Module

For speedup, Sort Module, Fitness Module, Crossover Module and Mutation Module operate in parallel.

Sort Module:

The block diagram of the Sort Module is shown in figure 6.

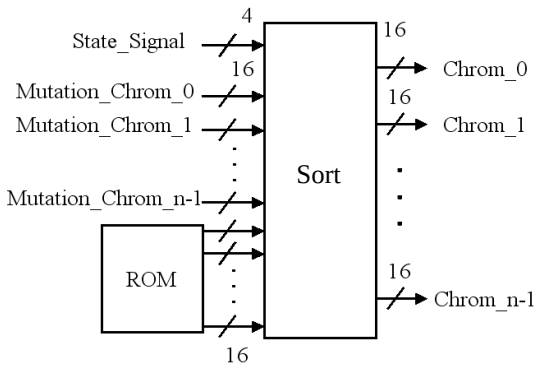

Figure 6: Sort Module

Memory Module operates whether reception from "Mutation_chrom" or generation of first chromosomes by "State_Signal". If the process is first generation, Memory Module generates first generation chromosomes form "ROM". For the rest, Memory receives new generation chromosomes from "Mutation_Chrom".

Fitness Module:

Fitness Module calculates fitness.

Fitness Module calculates the expressions below:

$$f(x) = |x^2 - 40{,}000|$$

x is binary input signal. Therefore, Fitness Module subtracts constant from multiplication result and outputs absolute value of the calculation result. If the calculation result is close to 0, it is a better chromosome. The optimized solution is 0. In this paper, we used 40,000 values as constant.

Selection Module:

The block diagram of Selection Module is shown in figure below:

IJDACR
ISSN: 2319-4863

IJDACR
International Journal Of Digital Application & Contemporary Research

International Journal of Digital Application & Contemporary research
Website: www.ijdacr.com (Volume 1, Issue 7, February 2013)

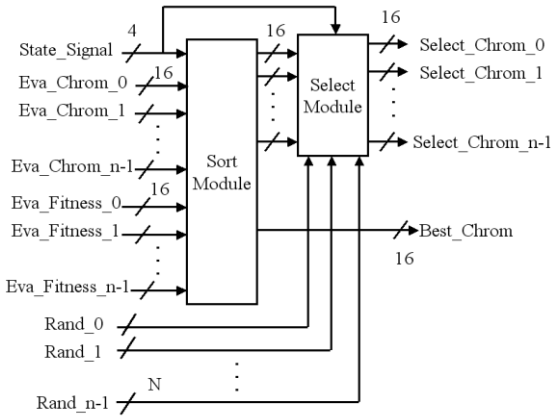

Figure 7: Selection Module

Selection Module consists of Sort Module and Select Module. Sort Module receives chromosomes Eva_Chrom and fitness Eva_Fitness by Fitness Module and sorts them in descending order of the low value of calculation result. As a sort method, we used insertion sort algorithm.

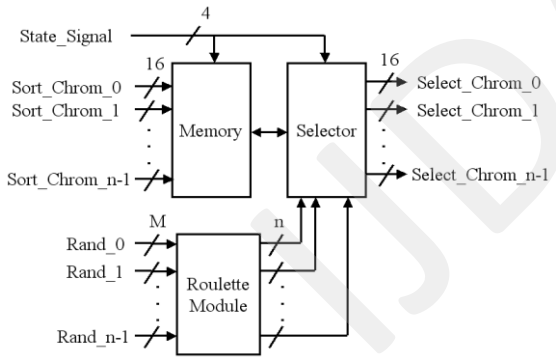

Figure 8: Select Module

The block diagram of Selection Module is shown in figure 8. Select Module stores away the sorted chromosomes in memory. Roulette Module operates roulette selection. It selects the individuals with high fitness. However, the individuals of low fitness may be selected with low probability. Selector decides individuals which become parents by chosen number by Roulette Module.

Crossover Module:

The block diagram of Crossover Module is shown in figure 9.

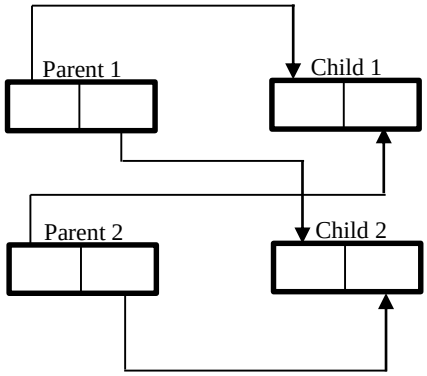

Figure 9: Crossover Module

Crossover Module generates next generation individuals by crossing chosen individual each other. The crossover method is one point crossover such as figure 1. The crossover point is decided by random number "Rand".

Decoder decides crossover point by random number. Selector crosses individuals each other or not by crossover point "Cross_ Point".

Mutation Module:

The block diagram of Mutation Module is shown in figure 10. Mutation Module changes the new individuals generated by crossover. Mask Generator generates mask "Mask" which decides mutation bit. Mutation Module performs the operation of "Mask" XOR "the individuals".

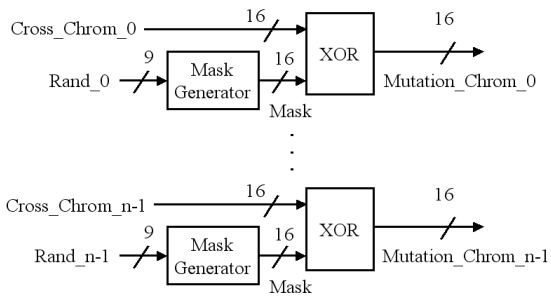

Figure 10: Mutation Module

IJDACR
ISSN: 2319-4863

IJDACR
International Journal Of Digital Application & Contemporary Research

International Journal of Digital Application & Contemporary research
Website: www.ijdacr.com (Volume 1, Issue 7, February 2013)

IV. Simulation and Results

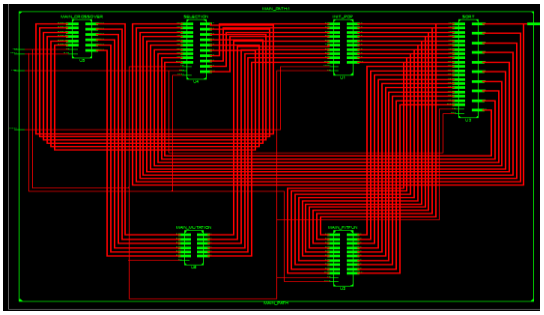

Figure 11: Main Module of GA

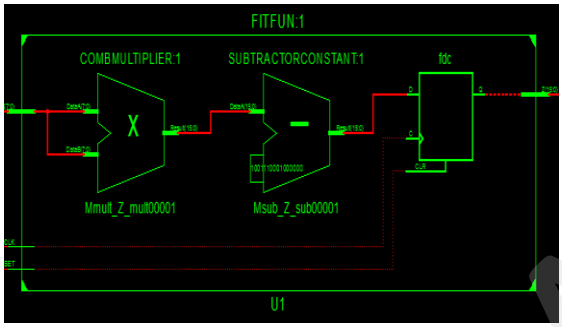

Figure 12: Fitness Function

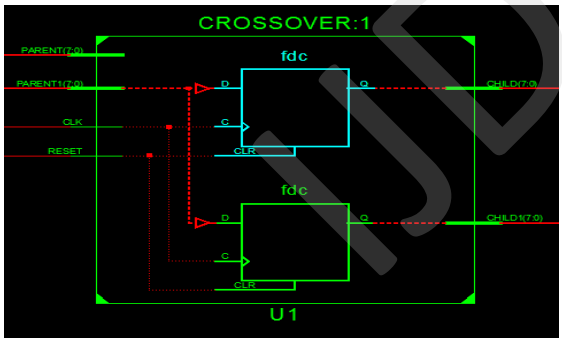

Figure 13: Crossover

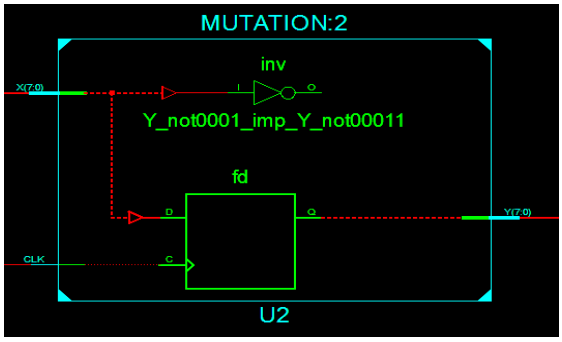

Figure 14: Mutation

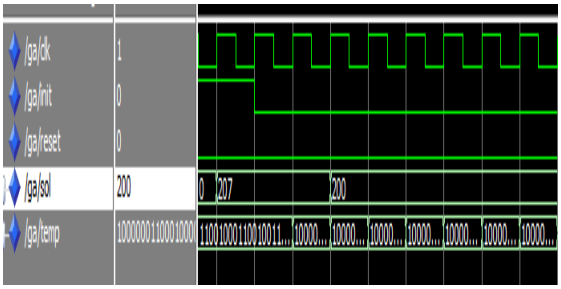

Figure 15: Simulation

Simulation and synthesis done by Modelsim and Xilinx ISE respectively. Figure 11 shows the main module of proposed GA. Figure 12 shows the fitness function. It optimize any input number by multiplying itself. When the output of fitness function reaches zero (0) then the square root of a particular number can be found. Here we have applied the optimization for finding the square root of 40,000.

Figure 15 shows the simulation of proposed work. 'init' is input and the output shows by 'sol'. We found, up to 3 iterations we get 207 and from fourth iteration we get 200 constantly. This is because of the output of fitness function reaches zero. And 200 is the answer after 20 generations which is square root of 40,000.

V. Conclusion

This paper presented a successful implementation of Genetic Algorithm using VHDL. We presented its architecture, design and evaluation. It is confirmed that the processor was effective in terms of calculating square root of a number by simulation results. The future work is evaluation for other functions.

Reference

[1] Stamenkovic, Z., Dahmen, H-Ch; Glaeser, U., "VHDL design validation by genetic manipulation techniques", IEEE, 2000.

[2] Matti T. Tommiska, "Area-efficient implementation of a fast square root algorithm", IEEE, 2000.

[3] Vedavathi.A, Meena. K.V & Gayatri.Malhotra, VHDL Implementation of Genetic Algorithm for 2-bit Adder", International Conference on Electronics and Communication Engineering, 2012.

[4] Tole Sutikno, Aiman Zakwan Jidin, Auzani Jidin and Nik Rumzi Nik Idris, "Simplified VHDL Coding of Modified Non-Restoring Square Root Calculator",

IJDACR
International Journal Of Digital Application & Contemporary Research

International Journal of Digital Application & Contemporary research
Website: www.ijdacr.com (Volume 1, Issue 7, February 2013)

International Journal of Reconfigurable and Embedded Systems, 2012.

[5] L. Davis ed., "Handbook of Genetic Algorithms", Van Nostrand Reinhold, 1991.

[6] G. Xianyue, L. Hongyan, and W. Shufeng, "A Dynamic Byte Encoding Genetic Algorithm for Numerical Optimization", IEEE Int. Conf. 3rd on Innovative Computing Information and Control (ICICIC'08), 2008.

[7] Peter Soderquist and Miriam Leeser "Division and Square Root: Choosing the Right Implementation", IEEE Micro, Volume 17, Number 14, July/August 1997, pp. 56-66.

[8] Masanao Aoshima, Akinori Kanasugi, "A Processor for Genetic Algorithm based on Redundant Binary Number", IEEE, Third International Conference on Convergence and Hybrid Information Technology, 2008