Volume 4, Issue 1, January 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Behavioral Modeling of Genetic Algorithm Processor using Model Sim

Hemant Kumar Bansal* *M.tech Stud.VLSI Tech .*
MMEC,MMU , India

Devesh Mahor[2] *Astt.prof.ECE*
Dept,MMEC,MMU, India

Abstract-Genetic Algorithm is one of the well-known optimization methods that belong to the class of Evolutionary Algorithms (EA). This method has been recognized to have successfully solved many problems in recent years, especially with respect to engineering and industrial problems. Even though it is one of the different type of EA, this method shares a lot of commonalities in the genetic operators, they use and the way they mimic natural evolution. In this paper, we present the behavioral modeling for the Genetic algorithm processor using Model Sim. It has been successfully synthesized and verified on a Model Sim 6.2c pro Field programmable gate arrays device (xc3s500e-5fg320).

Keywords: Genetic algorithm, Mutation, Selection, Fitness memory, Crossover, Field programmable gate array.

1. Introduction

Genetic Algorithm (GA) are evolutionary computational models based on Charles Darwin's theory of natural evolution based on the concept of the survival of the fittest. GAs are used to solve linear and non linear functions by exploiting areas through selection, mutation and crossover operators applied to chromosomes in the population [1], [2]. GA uses multiple individuals where each individual includes a chromosome representing a point in a search space of a given problem [3]. Genetic algorithm is very powerful technique to find approximate solution to search problems or patterns through application based on biological terms. Genetic algorithms provide a general approach for searching for global minima or maxima within a bounded and quantized search space. Since GA only requires a way to evaluate the performance of its solution guesses without any prior information, they can be applied generally to nearly any optimization problem. GA does not guarantee convergence nor that the optimal solution will be found, but do provide, on average, a "good" solution [6], [7]. GA is usually extensively modified to suit a particular application. As a result, it is hard to classify a "generic" or "traditional" GA, since there are so many variants. However, by studying the original ideas involved with the early GA and studying other variants, one can isolate the main operations and compose a "traditional" GA. An improvement to the "traditional" GA to provide faster and more efficient searches for GAS that does not rely on average chromosome convergence (i.e. applications which are only interested in the best solution). The "traditional" GA is composed of a fitness function, a selection technique, and crossover and mutation operators which are governed by fixed probabilities. The Block diagram of genetic algorithm processor is shown in fig 1.

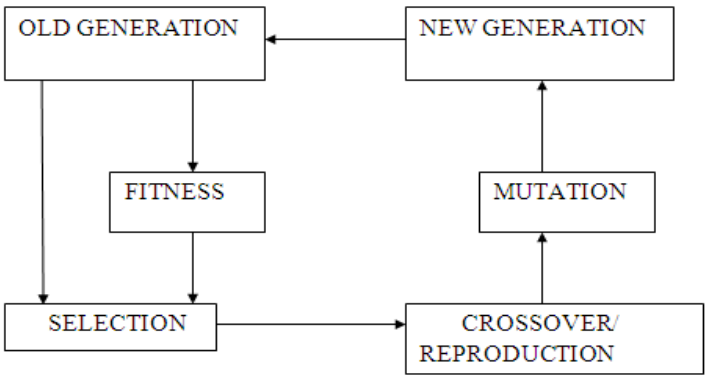


Fig 1 Block Diagram of Genetic Algorithm Processor

2. Genetic Operators

The three genetic operations are selection (or reproduction), crossover, and mutation. They are the core of the algorithm. The **selection** operator selects good chromosomes on the basis of their fitness values and produces a temporary population, namely, the mating pool. The selection operator is responsible for the convergence of the algorithm. The **crossover** operator is the main search tool. This is the **mutation** operation. The mutation operator is included to prevent premature convergence by ensuring the population diversity.

3. Flow Chart of Genetic Algorithm Processor

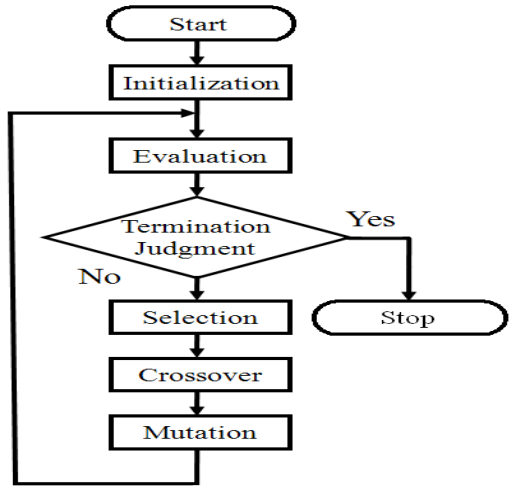


Fig 2 Flow Chart of Genetic Algorithm Processor

4. Why Genetic Algorithm

Genetic Algorithm is far better than conventional algorithm. Genetic algorithms are simple to construct and do not necessitate a significant amount of storage and making them a sufficient choice for an embedded system with an adequate processor and small fixed storage. Furthermore genetic algorithms are a good choice for embedded systems that gather data from sensors as the algorithms can provide decision-making based on this data [4], [5]. Genetic algorithm (GA) is one of the widely used techniques for constrained optimization. Performance of genetic algorithm can be improved with the introduction of some knowledge about the scheduling problem represented by the use of heuristics. Genetic algorithms use biologically inspired techniques such as genetic inheritance, natural selection, mutation and sexual reproduction (recombination or crossover). Some main reasons to use a genetic algorithm are:-

1. There are multiple local optima.
2. The objective function is not smooth (so derivative methods cannot be applied).
3. The number of parameters is very large.
4. It is better than conventional algorithm as it is more robust.
5. Unlike older algorithm system, the genetic algorithm does not break easily even if the inputs changed slightly, or in the presence of reasonable noise.
6. While performing search in large state-space, or multimodal state-space, or n-dimensional surface, a genetic algorithm offer significant benefits over many other typical search optimization techniques like – linear programming, heuristic, depth-first and breath-first.
7. Genetic Algorithms are good at taking large, potentially huge search space and navigating them, looking for optimal combinations of things, the solutions one might not otherwise find in a lifetime.

5. RTL View of Genetic Algorithm Processor

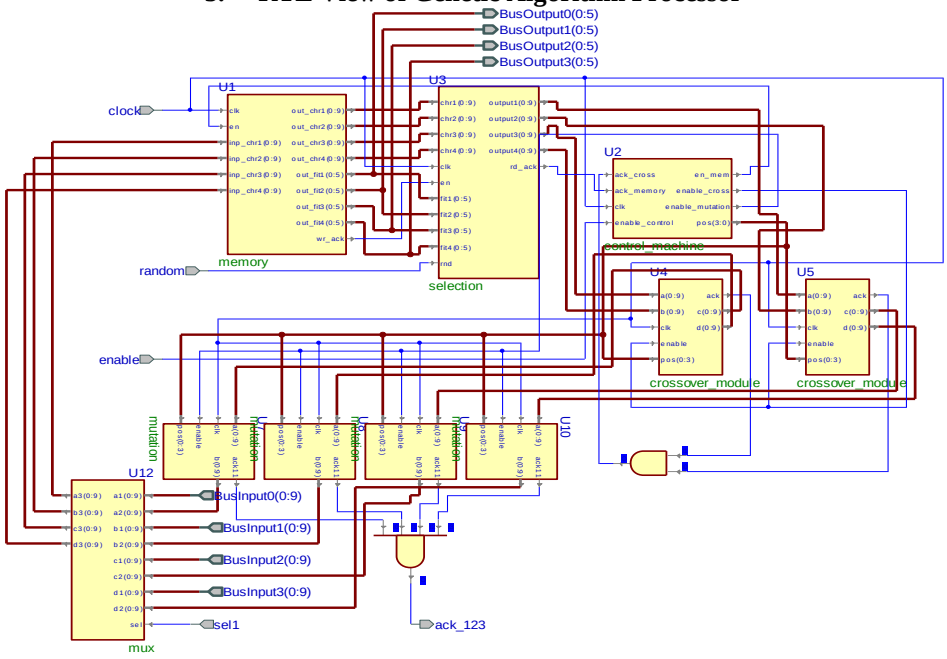


Fig 3 RTL view of genetic algorithm

6. Result and Discussion

Simulation Results of Function optimization using Random values

CASE 1

INPUT 0 = 201H RANDOM = 1
INPUT 1 = 203H
INPUT 2 = 207H
INPUT 3 = 20FH
----------OUT_PUT 0 = 205
OUT_PUT 1 = 205
OUT_PUT 2 = 205 OUT_PUT 3 = 205

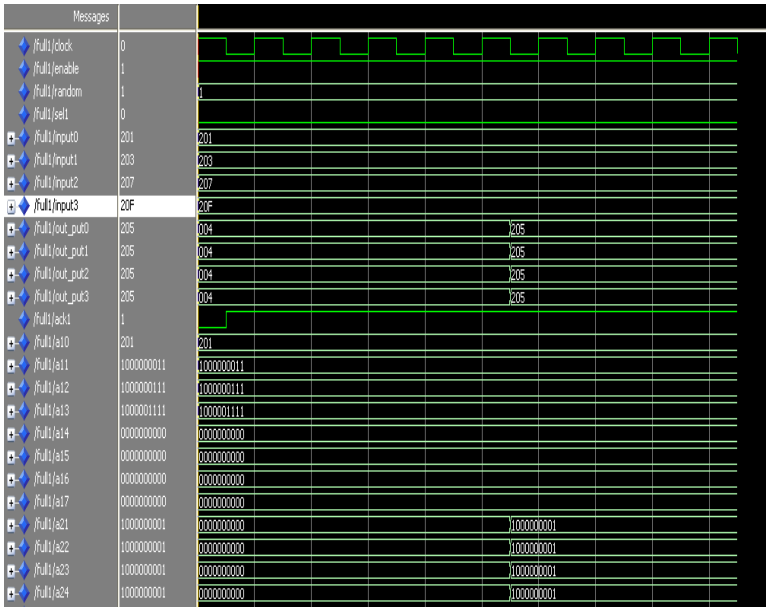
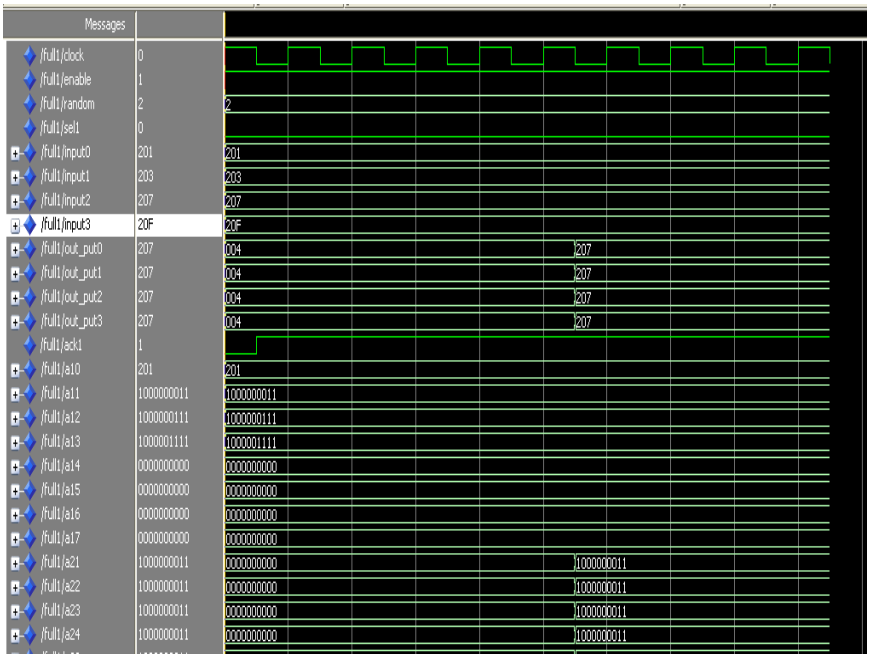


Fig 4 Simulation result of function optimization using random value 1

CASE 2

INPUT 0 = 201H RANDOM = 2
INPUT 1 = 203H
INPUT 2 = 207H
INPUT 3 = 20FH
----------OUT_PUT 0 = 207
OUT_PUT 1 = 207 OUT_PUT 2 = 207
OUT_PUT 3 = 207



CASE 3

INPUT 0 = 201H RANDOM = 3
INPUT 1 = 203H
INPUT 2 = 207H
INPUT 3 = 20FH
----------OUT_PUT 0 = 203
OUT_PUT 1 = 203 OUT_PUT 2 = 203
OUT_PUT 3 = 203

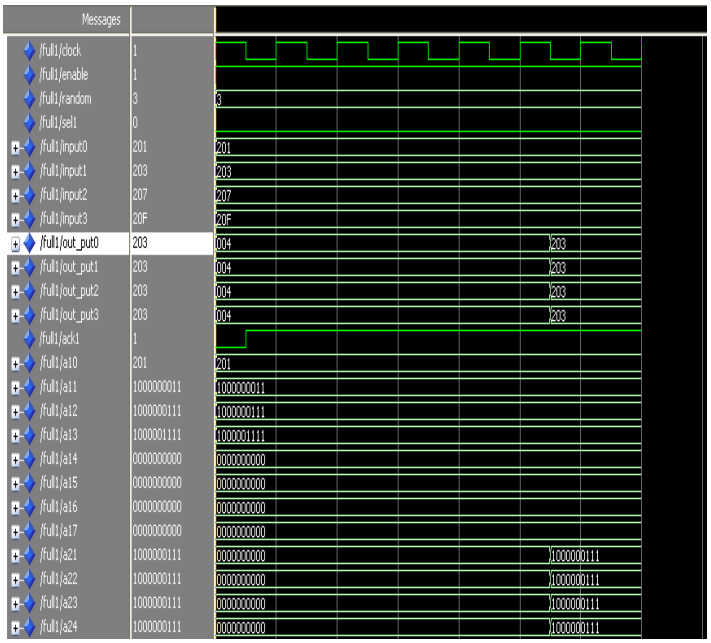


Fig 2 Simulation result of function optimization using random value 3

7. Discussion

1. It is observed that DC is better irrespective of function, threshold, and iterations. For the select nonlinear objective functions DC converges faster than single point, 2-point, or 3-point crossover mechanisms.
2. For linear and non-linear objective functions, we achieve maximum value using GA with DC that reaches up to 99.9999% of the actual maximum value given by the function.
3. The FPGA implementation has shown that the control unit for implementing a nonlinear (complex) function using 2-point, 3-point or DC employs exactly same number of slices.
4. For less complex linear functions, it is observed that the behavior of the DC mechanism is almost identical to that of single point crossover. This is in accordance with the fact that there may be virtually no need to employ the DC as single point crossover is sufficient enough.
5. From our observations, we conclude that dynamic crossover management is the way to future hardware accelerated genetic algorithms establishing substantial a performance improvement.

8. Conclusion

Genetic algorithms are significant methods to solve applications where there is practically no deterministic solution. However, these algorithms take more time to converge when implemented in software. We developed a genetic algorithm with dynamic crossover mechanism to exploit the inherent parallel nature. Dynamic crossover mechanism finds near maximum/minimum value of the selected objective function by dividing the entire population into blocks for parallel execution. It dynamically manages crossover points and migrates during runtime. Our results clearly indicate that algorithm outperforms other static approaches. We have shown that dynamic crossover management is well suited for parallelized versions by efficiently supporting migration. DC mechanism is implemented for hardware accelerated GA using FPGA. The outcome is faster convergence with limited memory resource. Currently, implementation supports linear and nonlinear objective functions, the evaluation are done using hardware complexity and convergence rate as the metric.

References

[1] Mohamed Sadek Ben Ameur, Anis Sakly and Abdellatif Mtibaa, "Implementation of Real coded Genetic Algorithms using FPGA Technology", 2013 10th International Multi-Conference on Systems, Signals and Devices (SSD) Hammamet, Tunisia, March 18-21, 2013.
[2] Godkin Andrey and Nonel Thirer "A FPGA Implementation of Hardware Based Accelerator for a Genetic Algorithm" 2010 IEEE 26-th Convention of Electrical and Electronics Engineers in Israel.

[3] Tatsuhiro Tachibana, Yoshihiro Murata, Naoki Shibata, Keiichi Yasumoto and Minoru Ito, "General Architecture for Hardware Implementation of Genetic Algorithm", 2006 IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06).

[4] C.G.Lopes and A.H.Sayed, "Distributed Adaptive Strategies: Formulation and Performance Analysis", in PROC. IEEE Int. Conference, Acoustics, Speech, Signal Processing, (ICASSP), Toulouse, Vol 3. France , May 2006.

[5] Hill, Seamus and O' Riordan, Colm. "Genetic Algorithms, their operators and the NK-model", NUI-IT-150601, Dept. of Information Technology, NUI, Galway, Ireland. 2001.

[6] Goldberg, D. E. "Genetic Algorithms in Search Optimization and Machine Learning", Addison-Wesley, 1989.

[7] L. Davis (Ed.), "Handbook of Genetic Algorithm", Van No strand Reinhold, New York, 1991.

