2581-4575
International Journal For Recent Developments in Science & Technology
Crossref
A Peer Reviewed Research Journal

IMPLEMENTATION OF GENERIC ALGORITHM USING VHDL ON FPGA

[1]NEERUGUTTA SUNITHA, [2]M.PRADEEP
[1]STUDENT, A1 GLOBAL INSTITUTE OF ENGINEERING AND TECHNOLOGY
[2]ASSISTANT PROFESSOR, A1 GLOBAL INSTITUTE OF ENGINEERING AND TECHNOLOGY

Abstract: Architecture. The development of a flexible very-large-scale integration (VLSI) for GA has been proposed in this paper. For the hardware architecture, we has develop on a random number generator (RNG), crossover, and mutation based on flexibility structure. This structure can dynamically perform to the 3 types chromosome encoding: binary encoding, real-value encoding, and integer encoding. The overall structures has been designed and synthesized by VHDL (VHSIC hardware description language), simulation by ModelSim program, and then implemented on FPGAs (Field programmable gate arrays). This hardware architecture that our design work very well flexible for the 3 groups problem examples: combinatorial optimization problems function optimal problems.

Keywords: Generic Algorithm, VHDL, FPGA

1 INTRODUCTION

Genetic algorithms (GA) are techniques used to find exact or approximate solutions to optimization and search problems its algorithmic structure is simple. In Figure 1 each of the block modules performs a simple operation: (i) the fitness module performs the evaluation of the chromosome, (ii) the sequencer module randomly selects the chromosomes, i.e. an aspect of the model under study, and passes them to the selection module, (iii) the selection module decides which of the sequenced module should advance, and (iv) the mutation and crossover modules mutate and mate the selected chromosomes. The need for hardware implementation of GAs arises from the vast computational complexity of problems that cause delays in the optimization process of software implementations. The speed advantage of hardware and its ability to parallelize, offer great advantages to genetic algorithms to overcome those problems. Speedups of 1 to 3 orders of magnitude were achieved when frequently used software routines were implemented in hardware with Field Programmable Gate Arrays (FPGAs). However, those implementations were focusing on solving one specific problem due to the hardware resources constraints This paper is organized as follows. Section II,describes basics of GA and classification of the chromosomes encoding. Section III,

International Journal For Recent Developments in Science & Technology
2581-4575
Crossref
A Peer Reviewed Research Journal

proposed GA hardware architectures. Section IV, discusses the simulation result. Section V, is conclusion

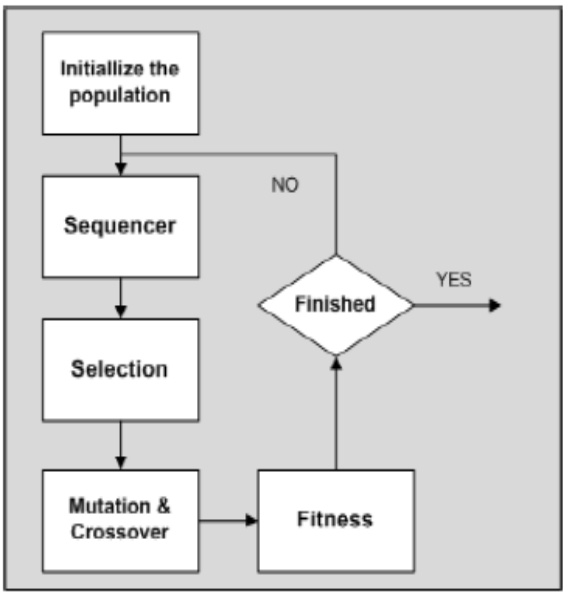

Fig1 Genetic Algorithm Flowchart

A. The Basics of GA

In general, the step of GA operations consists of 6 main steps: population initialization, fitness calculation, selection, crossover, mutation and termination judgment, is shown in Fig.2. In the beginning, the initial population of a GA is generated randomly. Then, the evaluation values of the fitness function for the individuals in the current population are calculated. And then, the 3 steps of GA operators: selection, crossover, and mutation are performed. Finally, the termination criterion is checked, and the whole GA procedure stops if the termination criterion is reached. Our design and develop hardware in the GA process conclude RNG operation, crossover operation, and mutation operation.

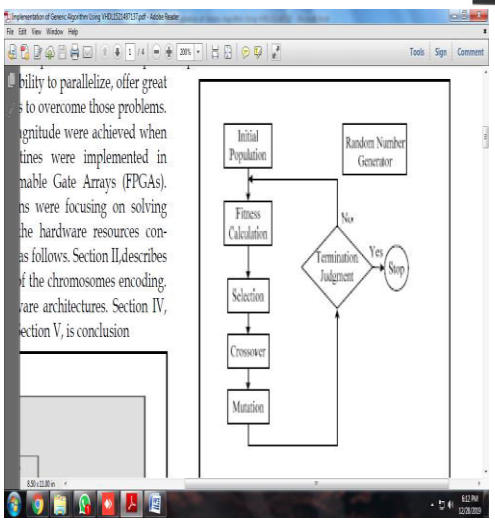

Fig2 Flow Chart of GA Process

B. Classifications of the Chromosome Encodings In this paper, we have classified the chromosome encoding according to the most prefer are 3 types compose of: binary encoding, real-value encoding, and integer encoding, that can describe are as follow:

- Binary encoding: In binary encoding, every chromosome is a string of bits only „0“ and „1“. For example “[11001110], [11100101]” Crossover operation: Used in selection of the genes from parent chromosomes and creates a new issue. The simplest way how to do this is to choose randomly some crossover point and everything before this point copy from a first parent and then everything after a crossover point copy from the second parent. Mutation operation: After crossover performed, mutation take place. This is to prevent falling all solutions in population into a local optimum of solved problem. Mutation changes randomly the new issue. Which, can switch a few randomly chosen bits from „1“ to „0“ or

International Journal For Recent Developments in Science & Technology
2581-4575
Crossref
A Peer Reviewed Research Journal

from „0“ to „1“. For example “[1 1 0 1 0 1 1] => [1 1 1 1 0 1 0]”

- Real-value encoding: Is the best used for function optimization problem. It has been widely confirmed that real value encoding performs better than binary encoding. In a real value encoding, every chromosome is a string of some value. The values can be anything connected to problem, from numbers, real numbers. For example “[1.23 2.47 3.21], [ABCDEF], [(right), (left)]” Crossover operation: All crossovers from binary encoding can be used. Mutation operation: Adding a small number to selected values is added (or subtracted). For example “(6 2.86 4.11 5.47) => (6 2.73 4.22 5.47)”

- Integer encoding: In integer encoding, every chromosome is a string of numbers, which represents number in a sequence. For example “[1 5 8 3 2 4 6 3 10], [10 2 5 9 6 4 1 3 8]” Crossover operation: One-point crossover is selected, till this point the integer is copied from the first parent, then the second parent is scanned and if the number is not yet in the offspring it is added. For example “(1 2 3 4 5 6 7 8 9), (4 5 3 6 8 9 7 2 1) => (1 2 3 4 5 6 8 9 7)” Mutation operation: Order changing, two numbers are selected and exchanged. For example “(1 2 3 4 5 6 8 9 7) => (1 8 3 4 5 6 2 9 7)” A basic idea in this work is to implement the hardware architectures of RNG (random number generation), crossover, and mutation. Because the 3 architecture are depend on the encoding operation. Difference encoding operation requires difference crossover and mutation. So, this hardware architecture design based on flexibility to the 3 types encoding for working together at a time. However the main drawbacks of hardware design are difference operation of crossover and mutation in each encoding. The main process of 3 hardware architecture are the users can choose any one of 3 type encoding according to the requirement of various GA applications, is shown in Fig 3.

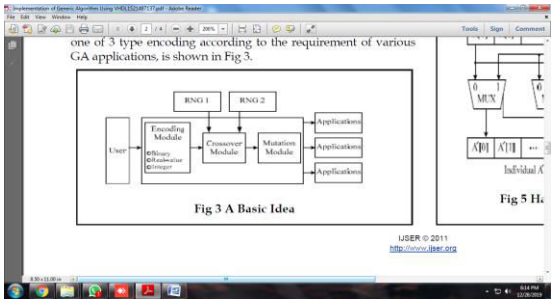

Fig 3 A Basic Idea

III. PROPOSED HARDWARE ARCHITECTURE FOR GA

In this paper, we have design and develop the 3 hardware architecture of the GA process, that concluding: random number generator module, crossover module, and mutation module, are as follow:

A. Random Number Generator Module

We have to take advantage of linear feedback shift register (LFSR) for random number generated. The operation of LFSR are generated by D flip-flop, is show in Fig. 4. And a first bit (X_0) to take XOR with a last-bit (Xn) represent feedback, that is repeat process many time. as following the equation (1).

$$X0(n+1) = Xn(n) + X0(n) \ldots\ldots\ldots\ldots (1)$$

International Journal For Recent Developments in Science & Technology
2581-4575
Crossref
A Peer Reviewed Research Journal

When X0(n +1) represent as data bit 0 at clock time (n +1), X_0(n) represent as data bit 0 at clock time (n), and X_n(n) represent as data bit n at clock time (n).

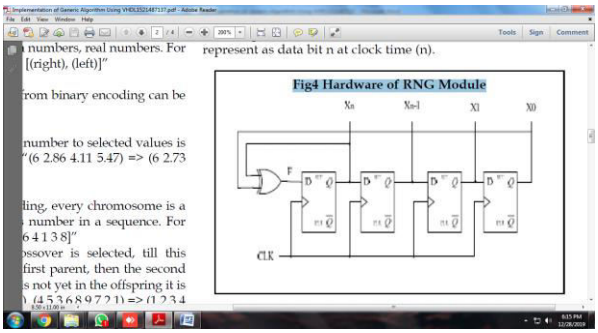

Fig4 Hardware of RNG Module

B. Crossover Module

The crossover module is used to perform the crossover operations on the two winner individuals, which are denoted A and B. The operator module offers, four crossover operations, including uniform, single-point, two-point, and cross-point. This crossover can use flexible to binary encoding, real-value encoding, and integer encoding. Users can choose any one of them according to their needs. The output chromosomes, which are denote A' and B', are then sent to the following mutation module, is show in Fig. 5.

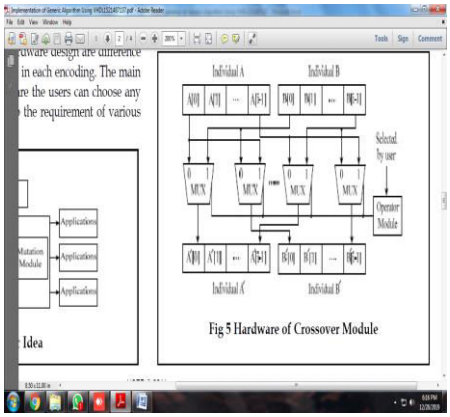

Fig 5 Hardware of Crossover Module

C. Mutation Module

The mutation module is used to avoid converging to the local optimization and instead locate the better solutions. A flexible mutation rate setting scheme is used. This mutation can use flexible to binary encoding, real-value encoding, and mutation encoding. User can easily choose an appropriate mutation rate according to their needs. In the design, the mutation operation is performed when the user-defined mutation rate exceeds the threshold, and it is also used to generate the new chromosome. Finally, the generated chromosome is into the population initialization, is show in Fig. 6

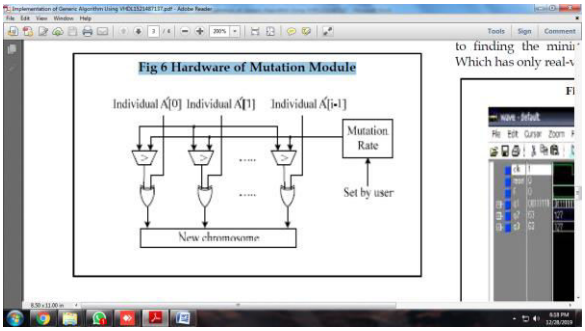

Fig 6 Hardware of Mutation Module

IV. SIMULATION AND RESULTS

The 3 groups problems, that is depend on the three types encoding have been experimental in this work. The group problems are as follow: Group#1, is combinatorial optimization problems, that compose of knapsack problem, minimum spanning tree problem, and set-covering problem. They have chromosome encoding to binary number. Group#2, is function optimal problems that is searching to max-min of the complex function, which has chromosome encoding as real value number.

International Journal For Recent Developments in Science & Technology
2581-4575
Crossref
A Peer Reviewed Research Journal

For example "find the maximum of a 1-d trigonometric function, and find the minimum of a 2-d Schubert function). Group#3, is path planning optimization problems This problem is about to find the optimal part from start point to final point. For example "find a optimal part of traveling salesman problem, and find a optimal part planning of a robot machine. The simulation result, we can used ModelSim program and VHDL code to design and synthesized. The simulation of RNG module is show in Fig. 7. The output can separate to three part: q1 represent binary output, q2 represent integer output, and q3 represent real-value output. The integer range -1 to 255, binary 8-bit, number of array equal to 36. For the VHDL code to design for RNG, we can used function in package "std_logic_unsigned" is conv_ integer (arg : std_logic_vector) return integer; and converse from integer to "std_logic_vector" can used function in package "std_logic_arith" is conv_std_logic_vector (arg : integer; size : integer) return std_logic_vector; and take 2s complement for conversion binary to real-number, and then conversion from real-number to integer respectively. So, the advantages of RNG module in this work is not use much time for design and reduce a cost for hardware implementation. The simulation of crossover module is show in Fig. 8. After the random number generated. The crossover operations are performed. In this example simulation for the binary encoding and real-value encoding can used. The VHDL code have to design for binary number = 8-bit, crossover point = 3 bit, integer n = 7. The simulation of mutation module is shown in Fig. 9. This simulation is an example of binary encoding and integer encoding. The VHDL code design is one-point mutation, integer n = 7, for binary number = 8 bit, and mutation point = 3 bit. The experimental results, we have compare our design with, which the flexible to finding the minimum-maximum of the complex function.

Which has only real-value encoding.

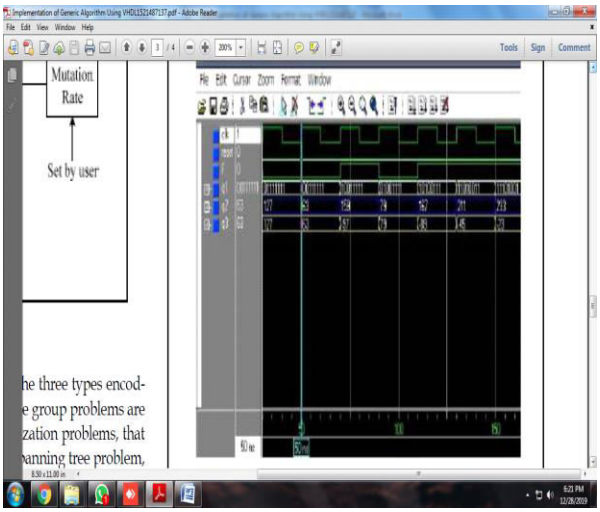

Fig 7 Simulation of RNG Module

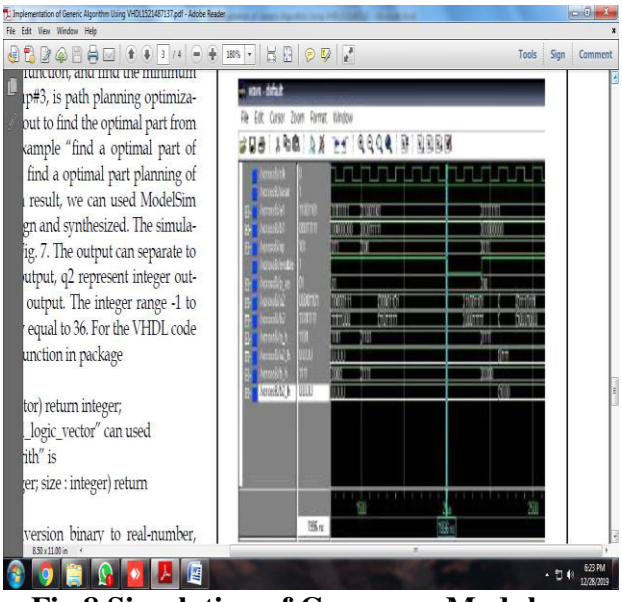

Fig 8 Simulation of Crossover Module

International Journal For Recent Developments in Science & Technology
2581-4575
Crossref
A Peer Reviewed Research Journal

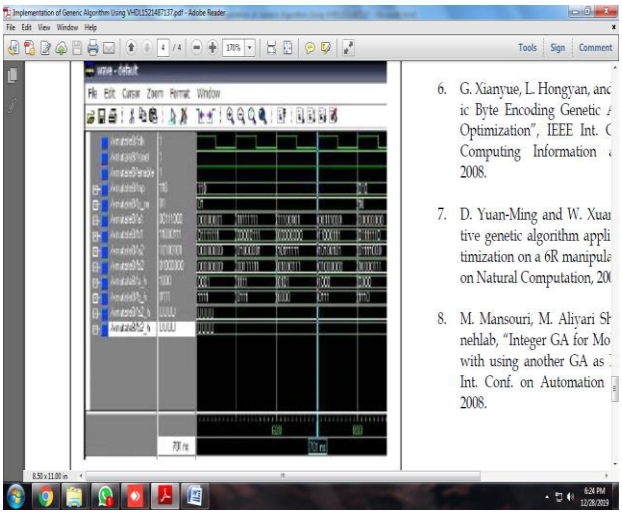

Fig 9 Simulation of Mutation Module

The 3 architectures have been designed using VHDL and then it is synthesized and verified. For the simulations, we used the ModelSim program and implementation will be on FPGAs. The GA requires very intensive computations to perform optimization. Hence, a dedicated VLSI implementation is necessary. All of most, a hardware GA requires very flexible to various applications. We have proposed development of a flexible hardware core for GA.

REFERENCES

- T. Tatsuhiro, M. Yoshihiro, S. Naoki, Y. Keiichi, and I. Minoru, "Flexiblen Implementation of Genetic Algorithm on FPGAs", IEEE Int.Conf., 2008.
- P. Vipapun and K. Pinit, "A High-Speed Hardware of the Geneticn Algorithm for Combinatorial Optimization Problem", in Proc.3rd IEEE Int. Conf. on Communication and Information Technology, vol. 2, September 2003.
- N. Shruthi and P. Carla, "Hardware Implementation of the Genetic Algorithm Modules for Intelligent System", IEEE Int. Conf., 2005.
- F. Pradeep, S. Hariharan, K. Srinivas, K. Didier, S. Adrian, Z. Ricardo, and R. Ramesham, "A Customizable FPGA IP Core Implementation of a General Purpose Genetic Algorithm Engine", IEEE Int. Conf., 2008.
- J. Yutana and C. Prabhas, "Cellular Compact Genetic Algorithm for Evolvable Hardware", IEEE Int. Pro. of ECTI-CON, 2008.
- 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.
- D. Yuan-Ming and W. Xuan-Yin, "Real-coded adaptive genetic algorithm applied to PID parameter optimization on a 6R manipulators", IEEE Int. Conf. 4th on Natural Computation, 2008.
- M. Mansouri, M. Aliyari Shoorehdeli, and M. Teshnehlab, "Integer GA for Mobile Robot Path Planning with using another GA as Repairing function",IEEE Int. Conf. on Automation and Logistic, September 2008.