Function Optimization Using Genetic Algorithm By VHDL

GJCST Computing Classification
B.5.2 C.m

[1]D.C. Dhubkarya , [2]Deepak Nagariya, [3]Jay Kumar,

Abstract- **This paper presents the work regarding the synthesis and implementation of a hardware genetic algorithm utilizing very high speed integrated circuit hardware description language (VHDL) for programming FPGAs. Genetic Algorithms were invented to mimic some of the processes observed in natural evolution. The idea with GA is to use this power of evolution to solve optimization problems. They are based on the principles of the evolution via natural selection, employing a population of individuals that undergo selection in the presence of variation-inducing operators such as mutation and recombination (crossover). we solved the problem with the help of hardware description language so it's take less time to find a result as compare to GA's because of HDL solve the problem by parallel processing. The present work deals with implementation and optimization of De jong's first function.**

Genetic algorithms need large memory banks to store the intermediate results and this has made the hardware implementation of GAs very inefficient but by using FPGA our task become simpler. Field-Programmable Gate Arrays (FPGAs) are flexible circuits that can be easily reconfigured by the designer. The program is written in VHDL and compiled with 32-Bit Microsoft Windows and implemented on a Spartan-3A FPGA from Xilinx.

Keywords- Fitness function, Crossover, mutation, random no, population, FPGA.

I. INTRODUCTION

A Genetic Algorithm is a search/optimization technique inspired by biological processes such as Natural Selection and Evolution. This project explores the application of such techniques to Parameter optimization problems [17]. A software framework was developed to store and manipulate data representations. Each data representation is a candidate solution to the problem being examined.

These candidates Solutions are grouped together into a family/generation. Starting from generation zero (Initial data), the algorithm iteratively processes each family allowing solutions to ~~–b~~reed", thus creating new candidate solutions. By discarding weak solutions and favoring the reproduction of strong ones the algorithm progressively refines each generation, leading to successive generations containing stronger solutions [10] we are using here VHDL for solving the problem of population control and these populations are randomly generated. So we use random bitgenerator here [7]. After thebit generation we need optimal solution result and also need fitness function. If you have above requirement than we select the best two functions and crossover between them and find the best result after finding the result we follow mutation operation if you find the less value of result as compare to fitness function then good otherwise you can follow these operation again and again.The paper is organized as follows. In section 2, an overview on Genetic Algorithm discussed in Section 3 previous work in brief. Section 4, presents the Hardware model Section 5 Simulation results and the paper is concluded in section 6.

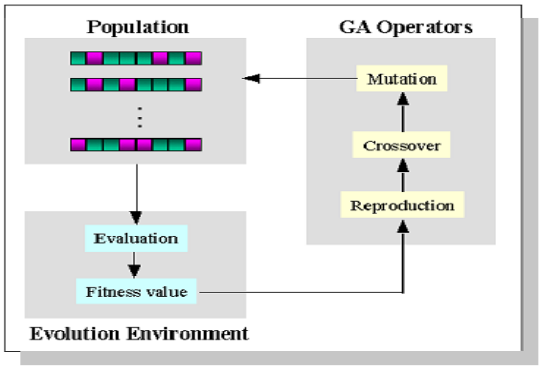

Figure (1) Evolution flow of genetic algorithm [13]

II. OVERVIEW OF GENETIC ALGORITHMS

Genetic algorithms (GAs) were invented by John Holland in the 1960s and were developed by Holland and his students and colleagues at the University of Michigan in the 1960s and the 1970s. In contrast with evolution strategies and evolutionary programming, Holland's original goal was not to design algorithms to solve specific problems, but rather to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation might be imported into computer systems [4]

Genetic Algorithms (GAs) are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics. As such they represent an intelligent exploitation of a random search used to solve optimization problems. Although randomized, GAs are by no means random, instead they exploit historical information to direct the search into the region of better performance within the search space. The basic techniques of the GAs are designed to simulate processes in natural systems necessary for evolution, specially those follow the principles first laid

[1]*D.C. Dhubkarya , BIET Jhansi ,India , dcd3580@yahoo.com*
[2]*Deepak Nagariya, BIET Jhansi ,India , deepaknagaria@gmail.com*
[3]*Jay Kumar, FET RBSAgra,India, jaykumar_1981@yahoo.co.in*

down by Charles Darwin of "survival of the fittest.". Since in nature, competition among individuals for scanty resources results in the fittest individuals dominating over the weaker ones. [11]

GAs simulates the survival of the fittest among individuals over consecutive generation for solving a problem. Each generation consists of a population of character strings that are analogous to the chromosome that we see in our DNA. Each individual represents a point in a search space and a possible solution. The individuals in the population are then made to go through a process of evolution. GAs is based on an analogy with the genetic structure and behavior of chromosomes within a population of individuals using the following steps:

A. Random Number Generation

A linear feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state. The only linear functions of single bits are xor and inverse-xor; thus it is a shift register whose input bit is driven by the exclusive-or of some bits of the overall shift register value. The initial value of the LFSR is Called the seed, and because the operation of the register is deterministic, the sequence of values produced by the register is completely determined by its current (or previous) state. Likewise, because the register has a finite number of possible states, it must eventually enter a repeating cycle. However, a LFSR with a well-chosen feedback function can produce a sequence of bits which appears random and which has a very long cycle. [7]

LFSR has two parts shift register & feedback function. A shift register is a device whose identifying function is to shift its contents into adjacent positions within the register. The feedback function is used in LFSR is XOR.[5]

B. Selection

Two chromosomes must be chosen from the population and recombined to produce a pair of new genomes in the new population for the next generation. Randomly choosing two chromosomes would be undesirable, as poor solutions would have an equal chance of being chosen as good solutions. Instead, a method such as roulette wheel selection is used. Roulette Wheel selection has been chosen for this application [14]. This means that the chance of an individual being chosen is proportional to its fitness. Individuals are not removed from the source population, so those with a high fitness will be chosen more times than those with a low fitness.

C. Crossover

With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.[3]

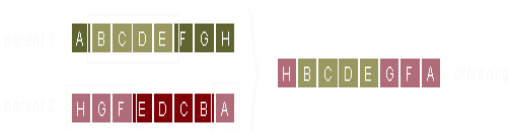

Figure 2 Crossover

D. Mutation

With a mutation probability mutate new offspring at each locus (position in chromosome)[3]

Figure 3 Mutation

III. RELATED WORK

There have been few reported studies on GA hardware implementations, one AHDL description has been announced in {10] but no performance estimations were made. In hardware description of GAPA system containing multiple FPGA chips and multiple digital signal processors was given. In our work we have implemented a GA in hardware using VHDL hardware description language and simulated the performance of the system on FPGA chip [10]

IV. HARDWARE IMPLEMENTATION

Our hardware consists of a Pentium microprocessor 4 which has high-speed PCI bus slots available. This is connected to Xilinx's 3 family of FPGA chip. The population resides in FPGA chip which are flexible RAM.

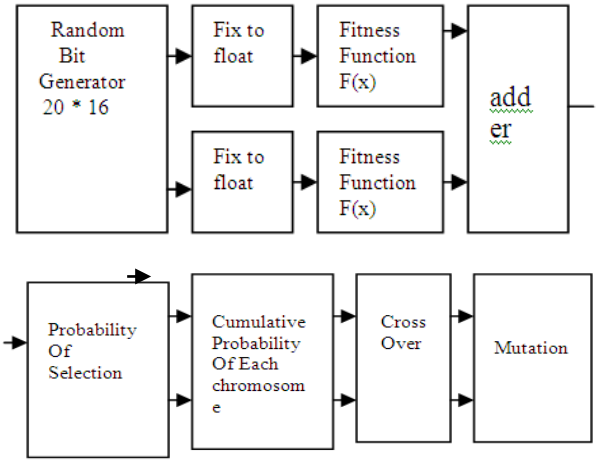

Fig 4 Block diagram of model

Initial Parameters

i. Crossover Rate:100%
ii. Crossover Type: 1 Pt.
iii. Mutation Rate: 2.5%
iv. Population Size: 10
v. Chromosome Length : 40 bit

A. Random Number Generation

In this block all the component of these blocks working to greater on the basic of clock pulse. Whenever we start the

FPGA kit. These all the process depend upon the clock and works as concurrently.
When the clock=0:- Random bit initialized the bit first bit between 0 and one and these bit passes to the solution string block after one clock delay.
When the clock =1:-New random bit generate for the next solution string and solution string passes to the next block and the population

B. Fitness Function

we have used the De Jong's function (1): since It is considered the easiest and simplest test function among De Jong's other functions [13]. It is also called ―TheSphere Model". It is a good example of a continuous, strong convex, unimodel function [15]. The structure of the first is defined as follows:

$$f_1(x) = \sum_{i=1}^{n} x_i^2$$

Fig 5 De Jong's functions

Where fi(x) is Fitness Function , & range for Xi is between 0 to 1.

Implementation

When the clock=0:-1st solution string selected and passed on to block 2nd.When the clock=1:-1st solution string pass to the next block Identifying genes*(X1 to X5) and solution string select a new solution string.
When the clock=2:-Identify genes (X1 to X5)1 pass to next block and convert this value in float form. And 2nd block identify the 2nd solution string and 1st block select 3rd string.
When the clock=3:-3rd block pass to the floating value to the next 4th block this block collect. Identify genes (X1 to X5)2 pass to next block and convert these value in float form. And 2nd block identify the 3nd solution string and 1st block select 4rd string.
When the clock=4:-in this clock 4th block pass to the value to 5th block .this 5th block calculate the some of fitness of each solution string. 3rd block pass to the floating value to the next 4th block this block collect. Identify genes (X1 to X5)3 pass to next block and convert these value in float form. And 2nd block identify the 4nd solution string and 1st block select 4rd string.All the Block works on the basis of clock pulse and each block have one latency time. So all block work after the single clock delay.

C. Selection

All the Block works on the basis of clock pulse and each block have one latency time. So all block work after the single clock delay.
When the clock=0:-fitness of the 1st solution string pass on to block 2nd (probability selection) and at the same clock 1st fitness goes to the fit sum block.
When the clock=1:-this block we have two in coming node 1st value come to the fitness block(2) and other value come to the fit sum (3)block and this block divide the value (fitness/fit sum).
When the clock=2:-this block calculate the cumulative vale of the fitness. it has single incoming node. The value comes from the 2nd block. And this time 2nd block divide the 2nd fitness.
When the clock=3:-4th block passes the cumulative value and this clock also involved to generate the new ten random number in between 0 to 1. And all previous block working in same nature in same clock.
When the clock=4:-in this clock value 6th block take the value from the 5th block and generate the parent

D. Cross Over

The algorithm for crossover operates as follows: The first chromosome is split in two at the crossover point. Both halves are then matched with the second Chromosome to find the longest common subsequence. Matching is done as follows: The first half is matched with the first character from the second chromosome, then with the first two characters and so on. The second half is matched with the last character from the second chromosomes, then with the last two characters and so on. The results of each match are stored in an array. The code in the fig implements the crossover part of GA.

```
begin
ncp<=CONV_INTEGER(cp);
ncp1:=ncp;
if (ncp1>38) then ncp1:=38; end if;
c1<=p1(39 downto ncp1) & p2(ncp1-1 downto 0);
c2<=p2(39 downto ncp1) & p1(ncp1-1 downto 0);
```

E. Mutation

Mutation is implemented by toggling randomly bit. Here mutation rate is 100% but can be varied easily. The code in the fig implements the mutation part of GA.

```
process(clk)
variable nm3:integer;
begin
nm3:=conv_integer(nm2);
if(nm3=40) then nm3:=39; end if;
sout<=sin(39 downto nm3+1)&(not sin(nm3))&sin(nm3-1
downto 0);
end process
```

V. Results

A top-down design methodology was adopted. A High-level VHDL model for the circuits was generated.[2] The logic was partitioned. Each part was re-described in a lower level description (RTL) required for the circuit synthesis, optimization and mapping to the specific technology by assigning current FPGA family and device. The resulting optimized circuit description was verified through extensive simulation. The proposed design was coded in VHDL. It

was functionally verified by writing a test bench and simulating it using ISE simulator and synthesizing it on Spartan 3A using Xilinx ISE 9.2i.[16]

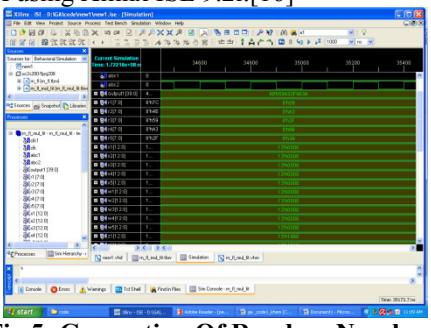

Fig 5: Generation Of Random Numbers

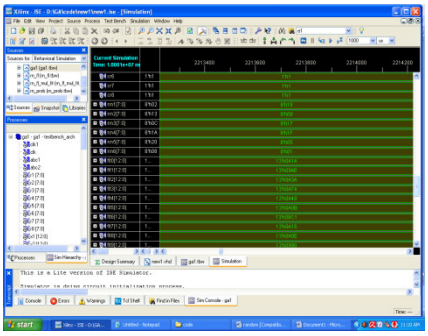

Fig 6: Optimisation Results

HDL Synthesis Report

Macro Statistics

# Adders/ Subtractors	: 1
32-bit adder	: 1
Counters	: 2
32-bit up counter	: 2
#Registers	: 82
13-bit register	: 40
2-bit register	: 1
32-bit register	: 1
40-bit register	: 22
8-bit register	: 10
9-bit register	: 8
#Comparators	: 6
32-bit comparator great equal	: 1
32-bit comparator less	: 1
8-bit comparator greater	: 4
# Multiplexers	: 2
40-bit 10-to-1 multiplexer	: 2
# Xors	: 83
1-bit xor2	: 73
1-bit xor3	: 10

The Xilinx Spartan 3A was adopted in our study as the features such as Suspend power-saving mode, high-speed I/O options, DDR2 SDRAM memory interface, commodity flash configuration support, and FPGA/IP protection using Device DNA Security. [16]. The Spartan-3A FPGA platform is a full feature platform of five devices with system gates ranging from 50K to 1.4M gates, and I/Os ranging from 108 to 502 I/Os, with density migration. The Spartan-3A FPGAs also support up to 576 Kbits of fast-block RAM with byte-write enable, and up to 176 Kbits of distributed RAM. Additionally, there are built-in multipliers for efficient DSP implementation and Digital Clock Managers (DCMs) for system level clock management function.

VI. Conclusion and Future Work

In this paper a We have studied the use of genetic algorithms in the optimization of Function F(x), the initial results are promising.

Other Genetic Algorithm operators could be implemented like, multi-point crossover, Partially Mapped crossover and different selection methods as well. The design can also be enhanced by incorporating a local search engine to create a hybrid memetic GA. The chromosome representation used in this project requires a relatively large amount of external memory to store the population and net list. Alternate chromosome representations can be explored in order to reduce the memory requirements. Furthermore, hardware/software co-design can be implemented and it can be compared with current implementation. Other real-time applications which require rapid and robust optimization can also be tackled with hardware based genetic algorithm.

VII. References

1) Chatchawit and Prabhas, —A Hardware Implementation of compace Genetic Algorithm", in Proceedings of the 2001 IEEE Congress on Evolutionary Computation, pp.624–629, Seoul, Korea, May 2001.
2) Douglas L Perry (2006), —VHDL Programming by Example", McGraw- Hill
3) G. Koonar S. Areibi M. Moussa, —Hardware implementation of Genetic Algorithms for VLSI CAD design", School of Engineering University of Guelph Guelph, Ontario,CANADA N1G 2W1.
4) Goldberg, D. E. (1989). —Genetic Algorithms in Search, Optimization & Machine Learning", Pearson Education, Inc
5) http://www.geatbx.com/docu/fcnindex.html #P74_1604.
6) http://www.obitko.com
7) Javad Frounchi Mohammad Hossein Zarifi Sanaz Asgari Far Hamed Taghipou —Design and Analysis of Random Number Generator for Implementation of Genetic Algorithms using FPGA"
8) . Microelectronic and Microsensor Research Lab, Faculty of Electrical and Computer Engineering, University of Tabriz,Iran
9) J. H. Holland, —Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence" MIT Pre Cambridge, MA, USA, 1992.

10) K. Skahill, VHDL for Programmable Logic, Addison Wesley, Reading, Massuchusetts, 1996.
11) Matti Tommiska and Jarkko Vuori,"Hardware Implementation of GA" Helsinki University of Technology, Otakaari Finland.
12) Mitchell Melanie," An Introduction to Genetic Algorithms", The MIT Press Cambridge.
13) Mustapha Abdulai, Inexpensive Parallel Random Number Generator for Configurable Hardware 2003
14) Nagham Azmi AL-Madi, Ahamad Tajudin Khader, De Jong's Sphere Model Test for A Social-Based Genetic", IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.3, March 2008
15) Russell Pinnington(1999) , —Implementation of Genetic Algorithm"MEng Cybernetics
16) T. Back. —Evolutionary Algorithms in theory and practice Evolution Strategies, Evolutionary Programming, Genetic Algorithms". Accessed 2008.
17) Xilinx Corporation, —ISE Manual", San Jose, CA.
18) [17] Zbigniew Michalewiez (1996)," Genetic algorithm + data structures = Evolution Programs",3rd Edition, Springer press.