"Enhancement of performance 32 bit RISC Processor using Genetic Algorithm"

Amit Yadav[1], Deepak Sharma[2]
Lord Krishna College of Technology

Abstract- Genetic algorithms (GAs) With 32 bit concept are used to solve search and optimization problems in which an optimal solution can be found using an iterative process with probabilistic and non-deterministic transitions. However, depending on the problem's nature, the time required to find a solution can be high in sequential machines due to the computational complexity of genetic algorithms. This work proposes a full-parallel implementation of a genetic algorithm on field-programmable gate array (FPGA) to perform as similar operation like as 32 bit RISC Processor. Optimization of the system's processing time is the main goal of this project. Results associated with the processing time and area occupancy (on FPGA) for various population sizes are analyzed. Studies concerning the accuracy of the GA response for the optimization of two variables functions were also evaluated for the hardware implementation. However, the high-performance implementation proposed in this paper is able to work with more variable from some adjustments on hardware architecture. The results showed that the GA full-parallel implementation achieved throughput about 16 millions of generations per second and speedups between 17 and 170,000 associated with several works proposed in the literature.

Keywords: *parallel implementation FPGA Genetic algorithms Reconfigurable computing.*

I. INTRODUCTION

1.1 Processor architecture and its fundamentals

A processor takes input information, processes that input and produces output result which gets stored inside the memory. This is the fundamental operation of a processor. The functional units of a processor are input output, memory storage, arithmetic logic unit and control unit. Information in a processor is in the form of data. In terms of technical language we can define this data in bits or bytes. The memory of the processor can be increased as per need.

Information in a processor is known as instruction which is given by an human operator. There is a particular language for the understanding of processor called as assembly language. It contains the instruction sets. A sequence of instruction is called as program. Processor fetches instructions that make up a program from the memory and performs the operations stated in those instructions. There are n no of programs written for a process.

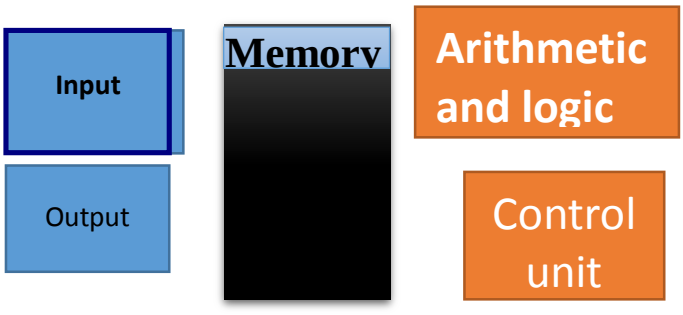


Figure 1.1: Overview of the RISC processor

1.2.1Memory:

Memory unit stores instructions and data. There are mainly two types of memory. The first is volatile and second is non volatile. Random Access Memory (RAM), is a volatile memory that stores information on an integrated circuit, whereas ROM (Read only memory) is a non-volatile memory and can last longer even after the power is off. The classification of memory is as follows-

1.2.2 Control unit:

All operations of processors are controlled by this unit. It actually generates timing signals which determines when the operation is going to take place. The interfacing of components with external elements is also done by this unit. It can be hardwired or micro programmed but hardwired controllers are fast. The functions of control unit are as follows-

- Control unit has communication among all components of the processor.
- It instructs the arithmetic logic unit that which type of operation is to be performed.
- It coordinates with peripherals of the processor.
- It directs all components to perform action.
- It regulates the flow of data between main memory and other units.

II. INTRODUCTION TO PIPELINING

This concept comes in to picture after decades of 80's.If we analyze any non-pipelined architecture, and then we found some hazards in it. There are five stages for processing any bit. The stages are namely Instruction fetch, decoder,

execution, memory access and write back. Now if any instruction is given to processor it has to pass through all stages to give an output. Therefore total no of clock cycles needed for16 bit instruction will be 16.Hence,we have to wait for a long time to get output and processor is called as slow processor. To speed up the work pipeline concept came in to picture. Now, the scenario will change.

Pipelining is a process where a microprocessor involves several stages of execution one after another in different segments. It is developed to reduce the total time for processing. These will in turn improves the speed of the processor. Using pipelining concept computer architecture allows next instructions to be fetched while executing the previous instructions. At the same time result is loaded to memory. We can increase the no of stages of pipelining as our application demands. For this we can understand this by simple example like there is a task T which has being assigned to me.Now,I can divide this task in to subtasks that is s1,s2 and s3.

A pipelined architecture is little more complex than non-pipelined. As we know that complexity always increase when design becomes small and speed becomes a major issue. We have to add some mechanism here, like some registers. Now in case of pipelining, in the first cycle t1 fetching is done and then its data is stored in register R1, thus the fetch register is free to fetch new bit. The data bit of R1 is now decoded by decoder in next cycle t2 and at the same time next data bit is fetched by fetch register. Similary, this bit of decoder is stored in register R2.This procedure is followed till write back. So total no of registers required are 5.Hence the total no of cycles are reduced. If 32 bits of instruction is there then total no of cycles required will be 11.Hence we get fast output in this case.

III. PROPOSED METHODOLOGY TO DESIGN OF 32 BIT PROCESSOR

The classical optimization techniques are useful in finding the optimum solution or unconstrained maxima or minima of continuous and differentiable functions. These are analytical methods and make use of differential calculus in locating the optimum solution. The classical methods have limited scope in practical applications as some of them involve objective functions which are not continuous and/or differentiable. Yet, the study of these classical techniques of optimization form a basis for developing most of the numerical techniques that have evolved into advanced techniques more suitable for today's practical problems.

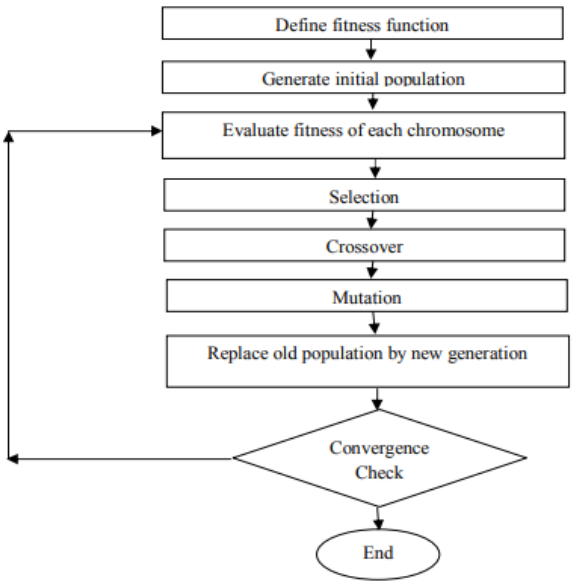
IV.PARAMETERS OF GENETIC ALGORITHMS

A number of parameters control the precise operation of the genetic algorithm. They are:

1. Crossover probability: It is the measure of how often crossover will be performed. If there is no crossover, offspring are exact copies of parents. If there is crossover, offspring are made from parts of both parent's chromosome. If crossover probability is 100%, then all offspring are made by crossover. If it is 0%, whole new generation is made from exact copies of chromosomes from old population. Crossover is made in hope that new chromosomes will contain good parts of old chromosomes and therefore the new chromosomes will be better.
2. Mutation probability: It is the measure of how often parts of chromosome will be mutated. If there is no mutation, offspring are generated immediately after crossover without any change. If mutation is performed, one or more parts of a chromosome are changed. If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed. Mutation generally prevents the genetic algorithm from falling into local extremes and helps in recovering the lost genetic material. Mutation should not occur very often, because then genetic algorithms would act as to random search.
3. Population size: It is the number of how many chromosomes are present in the population (representing one generation). If there are too few chromosomes, genetic algorithm has few options available for crossover and only a small part of search space is explored. On the counterpart, if there are too many chromosomes in one population then the speed of genetic algorithm slows down.

V. STEPS IN BASIC GENETIC ALGORITHM

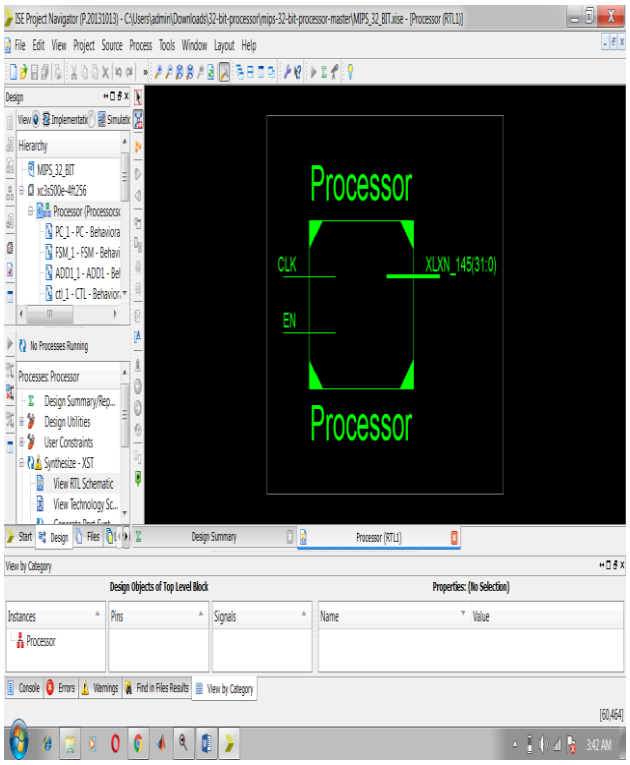
1. [Start] Define the fitness function f(x) according to the problem definition.
2. [Initialise] Generate random population of n chromosomes – each chromosome being the potential solution.
3. [Fitness] Evaluate the fitness f(x) of each chromosome x in the population.
4. [New population]Repeat the following steps to create the new population of chromosomes:
    a. [Selection] Select some parent chromosomes from a population according to their fitness to form mating pool.
    b. [Crossover] Mate the selected chromosomes as per given crossover probability to form new off-springs.
    c. [Mutation]Mutate new chromosomes as per given mutation probability.
    d. [Replace] Replace the old population of chromosomes with the new population.
5. [Convergence check] If the maximum number of generations is reached, then stop, and return the best solution. 6. [Loop] Go to step 3.



Basic flowchart of Genetic Algorithm

VI. RESULTS AND DISCUSSION

6.1 Processor pipelining in computer architecture (Implementation outcomes):



Fig(6.1) RTL 32 bit Processor pipelining.

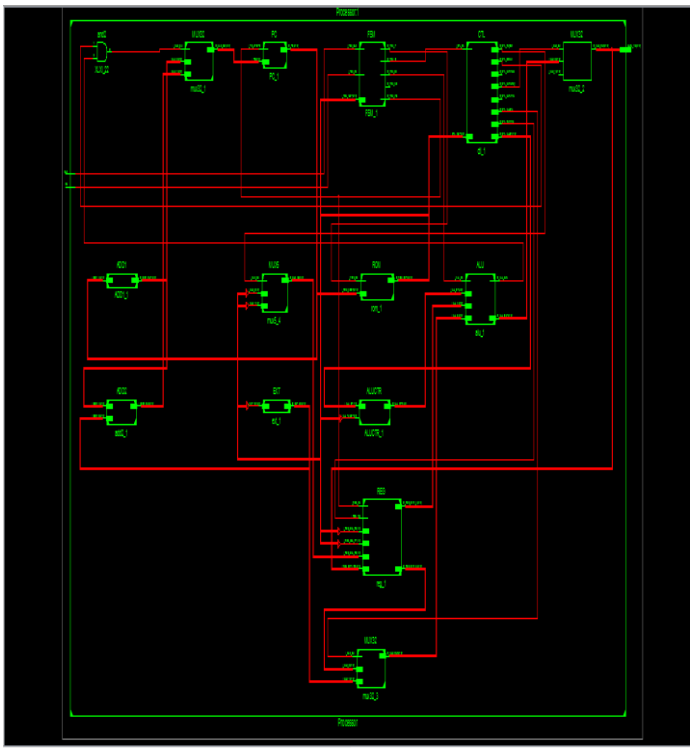


Fig (6.2) RAM, ROM, Accumulator, ALU, MUX, Adders all units of processor.

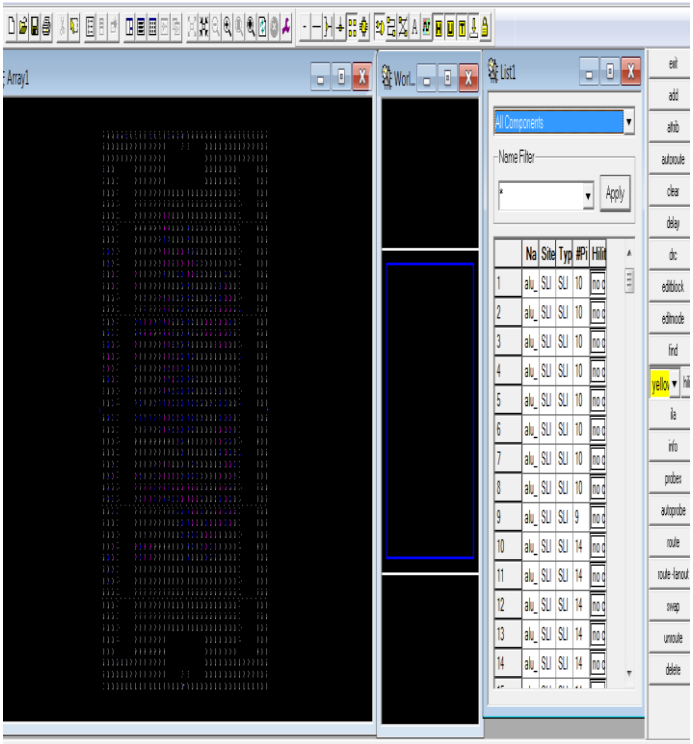


Fig (6.3) ALL unit routing.

Device		On-Chip	Power (W)	Used	Available	Utilization (%)
Family	Spartan3e	Clocks	0.000	37	–	–
Part	xc3s500e	Logic	0.000	2211	9312	24
Package	ft256	Signals	0.000	2232	–	–
Temp Grade	Commercial	BRAMs	0.000	4	20	20
Process	Typical	IOs	0.000	34	190	18
Speed Grade	-4	Leakage	0.081			
		Total	0.081			

Supply Summary		Total	Dynamic	Quiescent
Source	Voltage	Current (A)	Current (A)	Current (A)
Vccint	1.200	0.026	0.000	0.026
Vccaux	2.500	0.018	0.000	0.018
Vcco25	2.500	0.002	0.000	0.002

	Total	Dynamic	Quiescent
Supply Power (W)	0.081	0.000	0.081

Environment	
Ambient Temp (C)	25.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	0

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	31.1	82.5	27.5

Characterization

The Power Analysis is up to date.

(*) Place mouse over the asterisk for more detailed BRAM utilization.

Fig (6.4) Power dissipation of all unit.

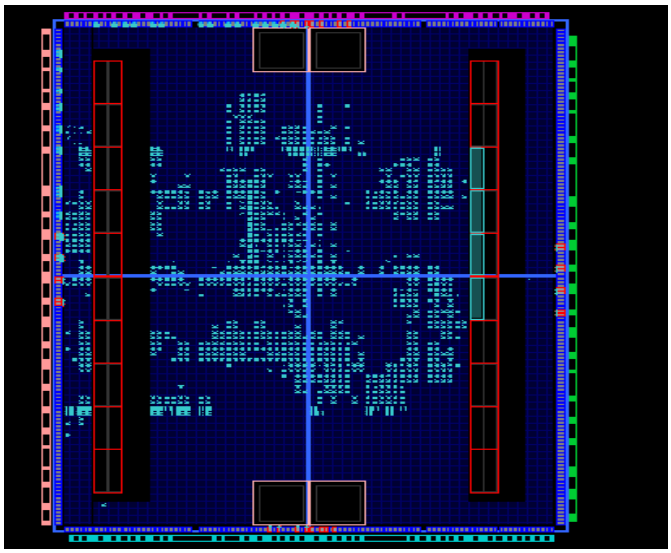


Fig (6.5) 32 Bit Processor.

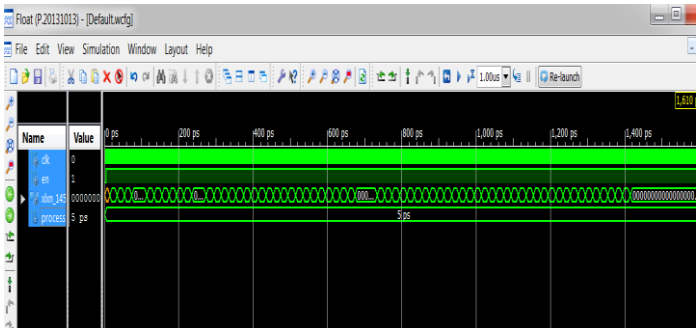


Fig (6.6) Processor instruction cycle.

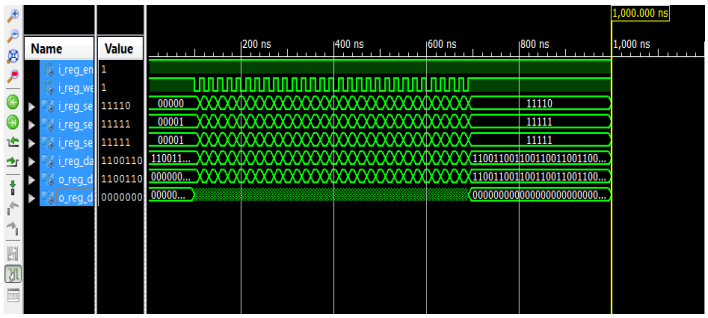


Fig (6.7) Internal Register outcomes

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	1,110	9,312	11%	
Number used as Flip Flops	15			
Number used as Latches	1,095			
Number of 4 input LUTs	2,207	9,312	23%	
Number of occupied Slices	1,131	4,656	24%	
Number of Slices containing only related logic	1,131	1,131	100%	
Number of Slices containing unrelated logic	0	1,131	0%	
Total Number of 4 input LUTs	2,212	9,312	23%	
Number used as logic	2,207			
Number used as a route-thru	5			
Number of bonded IOBs	34	190	17%	
IOB Latches	32			
Number of RAMB16s	4	20	20%	
Number of BUFGMUXs	24	24	100%	
Average Fanout of Non-Clock Nets	3.92			

Fig (6.11) Device Utilization summary.

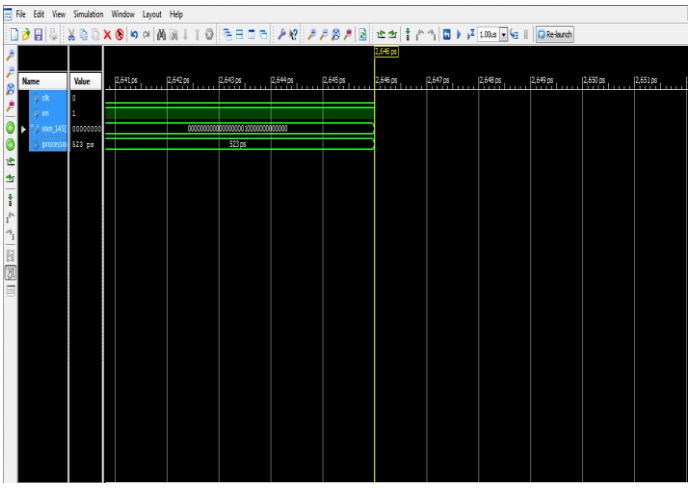


Fig (6.12) 32 bit Instruction cycle Time.

ENHANCEMENT REPORT

1. The first graph presents the change switching time with respect to variation in aspect ratio. It is important to see these results because it's the main thing which will affect the speed of the processor.

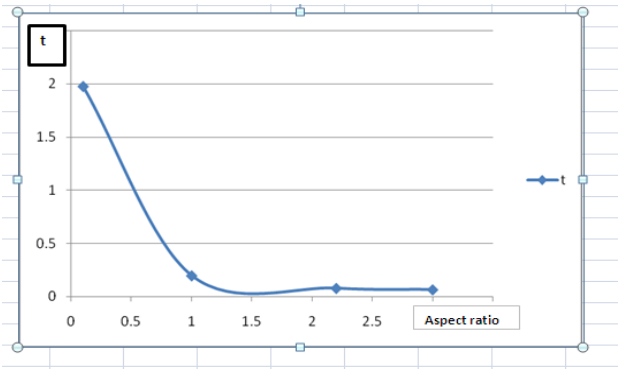


Fig 6.13 Graph between switching time & aspect ratio

From this graph it is very clear that as we increase value of aspect ratio the switching time decreases. Switching time is the time required to on or off the transistor. So that speed will increase because speed is indirectly proportional to time. For better results w=3l can be a good option.

2. This graph represents the relation between drain current and aspect ratio for three different materials Ge, Si and GaAs. Their motilities are different; other things are constant for them.

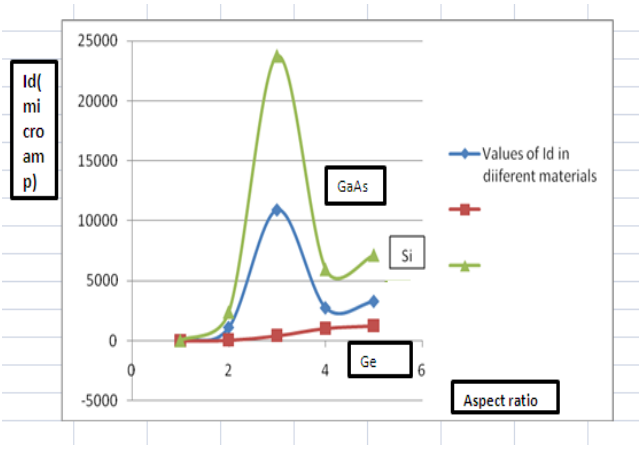


Fig 6.14 Graph represents Comparison of three elements based on mobility

From this graph we can observe that GaAs is the best because its curve reaches at max.Some results which are drawn from analysis at **w/l=3** are as follows;

- The threshold voltage for the processor is decreased.
- Power consumption naturally will reduce.
- The input impedance decreases to some content.
- Before the triggering was at5 V, now it is reduced to 3 V.
- RAM, ROM can be designed easily.

Applications

- ADC where we deal with very low voltage swings
- Microstrip antenna can be designed.
- High speed processor or controller can be designed.

RESULT COMPARISON & ANALYSIS

S.N.	Base paper	Proposed method
1	0.752ns	0.523ns

VII. CONCLUSION & FUTURE WORK

A pipelined architecture is a better option for us to support RISC processors. In this survey we observed the motivation for VLSI design. In comparison to microcontroller programming, this is better because program can be burned within the chip and erased easily, hence most of the customers are demanding for FPGA and designers are also moving in same direction.

Here we observed that w=3l is a better option for increasing the speed (Time<0.523ns) of the processor because drain current increases which in turn increases the switching capability of processor.

The speed will improve and to reduce the power consumption short transistors must be used. In this project we have shown the concept of pipelining through the comparison of different width to length ratio is done using the software XILINX'S. Optimization will take place in future as the demand will increase. Fabrication techniques can be improved for In future many things can be done to add features with this processor. Like more stages of pipelines can be added but at the same time hazards of pipelined should be taken care. The main idea was to optimize the design such as to increase speed of the processor; also the design uses a less amount of chip area.

REFERENCES

1.S.P.H. Alinodehi, S. Moshfe, M.S. Zaeimian, A. Khoei, K. Hadidi, High-speed general purpose genetic algorithm processor. IEEE Trans. Cybern. 46(7), 1551–1565 (2016)

CrossRefGoogle Scholar

2.M.S.B. Ameur, A. Sakly, Fpga based hardware implementation of bat algorithm. Appl. Soft Comput. 58, 378–387 (2017)

CrossRefGoogle Scholar

3.K. Chapman, Multiplexer design techniques for datapath performance with minimized routing resources, in Application Note: Spartan-6 Family, Virtex-6 Family, 7 Series FPGAs (2014)

Google Scholar

4.Y. Chen, Q. Wu, Design and implementation of PID controller based on FPGA and genetic algorithm, in 2011 International Conference on Electronics and Optoelectronics (ICEOE), vol. 4, p. V4–308. IEEE (2011)

Google Scholar

5.K. Deliparaschos, G. Doyamis, S. Tzafestas, A parameterised genetic algorithm IP core: FPGA design, implementation and performance evaluation. Int. J. Electron. 95(11), 1149–1166 (2008)

CrossRefGoogle Scholar

6.P. Fernando, H. Sankaran, S. Katkoori, D. Keymeulen, A. Stoica, R. Zebulum, R. Rajeshuni, A customizable FPGA IP core implementation of a general purpose genetic algorithm engine, in IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008 (IEEE, 2008), p. 1–8

Google Scholar

7.P.R. Fernando, S. Katkoori, D. Keymeulen, R. Zebulum, A. Stoica, Customizable FPGA IP core implementation of a general-purpose genetic algorithm engine. IEEE Trans. Evolut. Comput. 14(1), 133–149 (2010). https://doi.org/10.1109/TEVC.2009.2025032

CrossRefGoogle Scholar

8.M. Goresky, A. Klapper, Pseudonoise sequences based on algebraic feedback shift registers. IEEE Trans. Inf. Theory 52(4), 1649–1662 (2006)

MathSciNetCrossRefzbMATHGoogle Scholar

9.L. Guo, A.I. Funie, D.B. Thomas, H. Fu, W. Luk, Parallel genetic algorithms on multiple fpgas. ACM SIGARCH Comput. Arch. News 43(4), 86–93 (2016)

CrossRefGoogle Scholar

10.L. Guo, A.I. Funie, Z. Xie, D. Thomas, W. Luk, A general-purpose framework for FPGA-accelerated genetic algorithms. Int. J. Bio-Inspir. Comput. 7(6), 361–375 (2015)

CrossRefGoogle Scholar

11.J.H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence (U Michigan Press, Ann Arbor, 1975) zbMATHGoogle Scholar

12.N. Instruments, Understanding parallel hardware: multiprocessors, hyperthreading, dual-core, multicore and FPGAs (2011). http://www.ni.com/tutorial/6097/en/

13.L.M. Ionescu, A. Mazare, A.I. Lita, G. Serban, Fully integrated artificial intelligence solution for real time route tracking, in 2015 38th International Spring Seminar on Electronics Technology (ISSE) (IEEE, 2015), p. 536–540Google Scholar

14.Y. Jewajinda, P. Chongstitvatana, Hardware architecture and FPGA implementation of a parallel elitism-based compact genetic algorithm, in TENCON 2009-2009 IEEE Region 10 Conference (IEEE, 2009), p. 1–6

Google Scholar

15.J.R. Koza, Genetic evolution and co-evolution of computer programs. Artif. Life II(10), 603–629 (1991)Google Scholar