VOL. 10, NO. 11, JUNE 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences


www.arpnjournals.com

OPTIMIZING GA OPERATORS FOR SYSTEM EVOLUTION OF EVOLVABLE EMBEDDED HARDWARE ON VIRTEX 6 FPGA

Ranjith C., S. P. Joy Vasantha Rani, Priyadharsheni B., Medhuna Suresh and Madhusudhanan M.
Department of Electronics Engineering, MIT Campus, Anna University, Chennai, India
E-Mail: ranjith.kmct@gmail.com

ABSTRACT

The paper aims to provide an idea of the genetic algorithm parameters and its importance in the evolution of circuits through embedded evolvable hardware. Evolvable Hardware is an integration of evolutionary algorithms with programmable devices. A Genetic Algorithm fused into the soft processor of a Field Programmable Gate Array is termed, Evolvable Embedded Hardware. The system has the ability to converge to a solution faster due to the evaluation in a single device, when compared to the conventional evolvable hardware structure. An insight into the genetic algorithm and optimization of genetic parameters for design of combinational circuits is discussed. An experimental model for a 2 bit adder for different genetic parameters is validated to demonstrate the systematic evolution of evolvable embedded system hardware. This experimental setup is carried out on Virtex 6 (XC6VLX240T-1FFG1156) ML605 Evaluation Kit FPGA using the Xilinx Platform Studio 14.6 tools.

Keywords: evolvable hardware, evolutionary algorithm, genetic algorithm, evolvable embedded hardware, FPGA, VLSI.

1. INTRODUCTION

Conventional VLSI design methods using the Hardware Description Languages (HDL) are sometimes inadequate as the system complexity increases. The complexity of circuits has made design through human intervention all the more complicated. As the design complexity increases exponentially it becomes difficult to optimize the system in terms of speed, area and power. To overcome the hurdle of design ability and to reduce the human interference in the design we go in for a technique which is known as “Evolvable Hardware” or EHW. EHW was a concept introduced by Hugo De Garis in 1992 which was an integration of evolutionary algorithms with programmable devices [2]. In EHW, the circuit is designed and implemented in a programmable device by adaptively changing an algorithm which uses the Darwinian principle of natural selection [1]. The evolutionary algorithms include genetic algorithms, genetic programming, evolutionary computing, and so on, whereas the programmable devices are preferably Field Programmable Gate Arrays (FPGA). The benefits of implementing the circuits through evolutionary algorithms are – reduced complexity, smaller circuits, better performing and efficient circuits, and the user can bring about new ideas and innovations to the design [1]. The scope of this paper is limited to the integration of Genetic Algorithms (GA) with FPGAs in EHW.

The integration of GAs with FPGA in EHW is with regard that the configuration bit strings to the FPGA are the chromosomes of the GA as shown in Figure-1. An initial population of chromosomes for a target solution are evaluated for their fitness. The chromosomes are ranked based on their fitness and undergo genetic operations like crossover, mutations and selection. The final fit chromosomes are encoded as configuration bits of the FPGA, to form the new evolved circuit. A best fit solution is obtained after undergoing several iterations, to obtain the final optimized circuit. A new circuit configurations are evolved at every iteration based on the fitness probability. A fitness function is designed, such that the GA can autonomously find the best solution for the design, to be implemented in the FPGA. FPGA based EHW can be classified based on evaluation of the solutions - extrinsic evolution and intrinsic evolution as shown in Figure-1. In extrinsic evolution the development of circuits uses a simulation approach of determining the best evaluation and the solutions are implemented in the device. In intrinsic evolution, each candidate solution is directly mapped and implemented in the target device. The latter offers better accuracy of self evolved circuits [3]. In conventional works on EHW, the GA operations were conducted out in computer or workstations, which would make the system robust and slow. In recent works the GA operation is performed on the same FPGA chip either by hardware or by software in a dedicated core. The paper discusses this concept, where the GA operation is performed on a soft core processor, known as MicroBlaze, on the Virtex 6 FPGA. This type of evolution of circuits is termed as Evolvable Embedded System [4].

The concept of the Evolvable Embedded System is - an evolutionary algorithm is utilized to dynamically modify some of the system components, in order to adapt the behavior of the system to a changing environment [4]. A 32 bit MicroBlaze soft processor computes the GA for the evolution of a target system. The best fit chromosomes from the GA, configure the FPGA to give birth to an optimized or novel system. The process of evaluation of the best fit chromosomes, and evolution of the system takes place in the same FPGA. An evolvable embedded system architecture for a 2 bit adder circuit using the intrinsic mode of evolution is discussed, changing the hardware based on the fitness evaluation from the GA. This evolvable embedded hardware architecture was implemented on Virtex 6 (XC6VLX240T-1FFG1156) ML605 Evaluation Kit. GA results displayed on a PC.

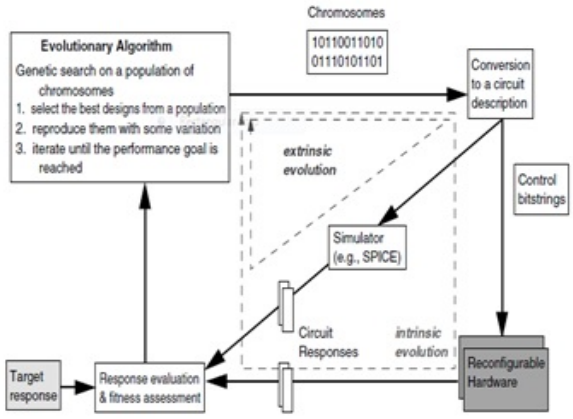

Figure-1. Basic structure of EHW.

The rest of this paper is organized as follows. Section II gives a detail study of GA and its parameters. Section III describes the modeling of an evolvable embedded system. Section IV gives a complete system design to implement a 2 bit adder. Section V discusses the results with Section VI giving a brief conclusion of the paper.

2. GENETIC ALGORITHM

The Evolutionary Algorithm (EA) brings about a hypothesis that a population of individuals exists in an environment with limited resources, and the competition for these resources causes selection of fitter individuals that are better accommodated to the environment [1]. Genetic Algorithm is an EA developed by John Holland in 1970's and popularized by David Goldberg. They are adaptive heuristic search algorithm based on the principle of the Charles Darwin theory of natural selection and genetics. GA is capable of solving a myriad of design parameters and multiple design goals, thus finding an optimal combination [6]. With the advent of the evolvable hardware concept, the role of GA became popular in the field of VLSI design. The aim of the algorithm to find an optimum solution to the problem was utilized here, to evolve circuits that may be optimized in area, speed or power. It can also lead to novel circuit structures compared to the manual design. A detail study of the GA and its characteristics is explained below.

a) GA terminology

A brief description of the terms associated with the GA is described:

- Population: it is a collection of several alternative solutions to the given problem.
- String or Chromosomes: it is the individual in the population.
- Genes: it is the individual characters in the string.
- Genotype: the bit string (chromosomes) that provides a possible solution.
- Phenotype: the genotype encoded into a physical structure
- Population size: it determines the amount of information stored.
- Fitness function: it is the user defined problem specification.

b) GA characteristics

There are five main characteristics of GA [1][6], which are problem dependent. Depending on the problem definition the characteristics differ in their approach.

1) Representation: it is a mode of representing the individuals in an optimized way, to store the representations. The representation can be either binary or real-valued representations. This paper deals with only binary representations

2) Selection: defined as selecting fit candidates from the pool, to pass on their genes to future generations. Different types of selection processes are - Truncation selection, Roulette wheel selection, Tournament selection, Neighborhood selection.

3) Mutation: genes are randomly altered in hopes of bringing out new properties into the next iteration.

4) Fitness function: gives an intuition of how well the individual is and depends on the problem.

5) Survivor decision: it is the survival of the best individuals, also known as Elitism factor. In general, the overall best individual is stored as a different individual, but they are not used during operations.

c) GA operators

The basic GA operators are crossover, mutation and selection, which establish the main algorithm, whereas the population and fitness function can be seen as external entities. Both crossover and mutation are probabilistic operations and their frequencies of occurrence are controlled by predefined probabilities. As a crossover plays the key role in improving the solution, it is assigned a high frequency of occurrence. The frequency of occurrence of mutation is kept fairly low, to prevent the GA from producing a large number of random solutions.

1) Crossover: recombination of genetic material of the parent to form one or more offspring, by preserving some of the useful traits of the parents. The goal is to generate new chromosomes that are more fit than their ancestors, thereby leading to the overall convergence of the population. There are many ways of performing crossover - one-point, two-point, or uniform crossover is used with binary encoding.

One point crossover: a random position in the chromosome is chosen. Child 1 is head of chromosome of parent 1 with the tail of chromosome of parent 2. Child 2 is head of chromosome of parent 2 with the tail of chromosome of parent 1 as shown in Figure-2.

Two point Crossover: two random positions on the chromosome are chosen. The genes at the head and the tail, of a chromosome are always split and then recombined as shown in Figure-3.

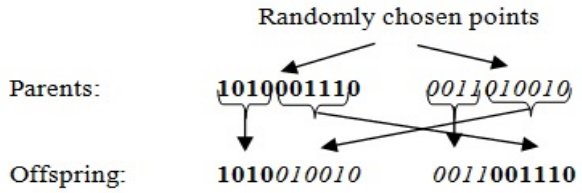

Figure-2. One point crossover.

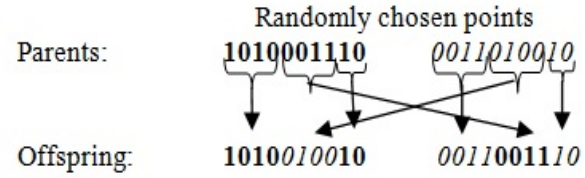

Figure-3. Two point crossover.

Uniform Crossover: a random mask is generated. The sequence of the mask determines the crossover from parent 1 and parent 2. Bit density in mask determines how much material is taken from the other parent as shown in Figure-4.

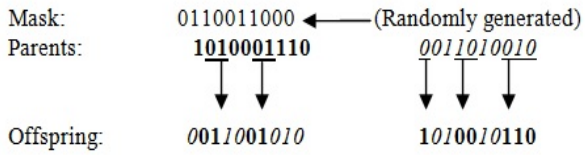

Figure-4. Uniform crossover.

Mutations: it randomly changes the bit of an offspring after crossover. Mutation is treated as supporting operator for the purpose of restoring lost genetic material. Bit flip mutation is the most common mutation operator for binary- encoded GAs. This is realized by simply inverting one or more bits in the chromosome string based on the probability of mutation as shown in Figure-5.

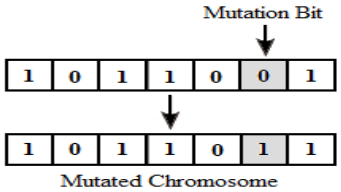

Figure-5. Mutation operator in binary representation.

Selection: different selection methods are available, but the most commonly applied methods are roulette- wheel, tournament and ranking. In the Roulette - wheel selection each individual's probability of being selected in the next population is proportional to its fitness value. Rank selection involves ranking the individuals from 'best' to 'worst' on the basis of their measured fitness values. In Tournament selection, a group of individuals is chosen iteratively by holding a tournament and the one with the best fitness value is chosen until it is filled with a predetermined number of individuals. This can be avoided by ensuring that a number of individuals deemed to be the best are always passed on to the next generation unchanged. This method is called elitism and it often increases the convergence speed at the expense of a risk of getting stuck around the so-called elite solutions.

d) GA flow

The complete flow graph of the GA program is illustrated in Figure-6. The flowchart can be explained as follows.

Step-1: Create an initial population of random solutions (chromosomes) by some means.

Step-2: Assess the chromosomes for fitness using the criteria imposed on the required solution and create an elite set of chromosomes by selecting a number of chromosomes that best satisfy the requirements imposed on the solution.

Step-3: If the top-ranking chromosome in the elite set satisfies fully the requirements imposed on the solution, output that chromosome as the required solution, and stop. Otherwise, continue to Step-4.

Step-4: Apply crossover between pairs of chromosomes in the elite set to generate more chromosomes and subject certain chromosomes chosen at random to mutations, and repeat from Step-2.

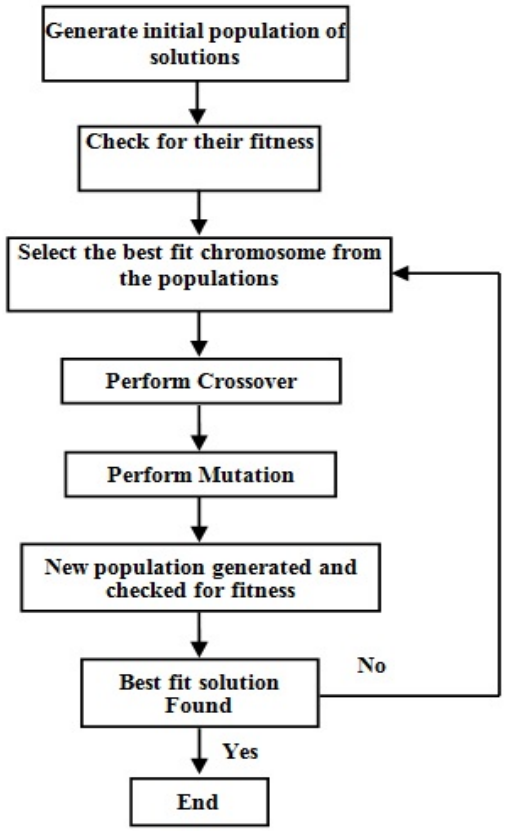

Figure-6. Flowchart of GA.

3. STRUCTURE OF EVOLVABLE EMBEDDED HARDWARE

The architecture uses the concept of the Virtual Reconfigurable Architecture (VRA) core [12] [13]. Here the VRA is modeled as an IP core using HDL. This hardware description of the architecture was layered over the reconfigurable chip, to implement the evolutionary structure. The main idea of this concept is that the designer has the flexibility in modeling the GA program to produce configuration bits to program the FPGA. GA program was fused in the MicroBlaze soft processor, where the evaluations are displayed on the PC through a UART peripheral. The complete system is housed in the ML605 FPGA board. The complete structure is as shown in Figure-7.

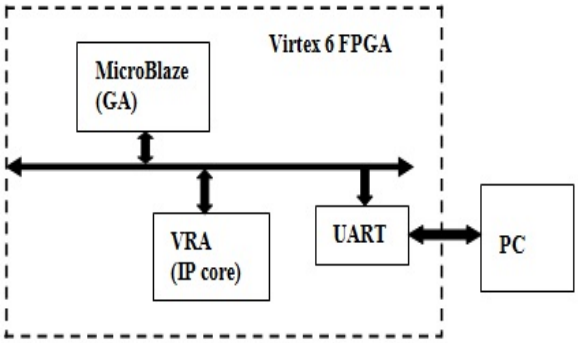

Figure-7. Structure of evolvable embedded system.

VRA is described in HDL, and is considered as a second reconfigurable layer on upper side of the FPGA [12]. The main advantage of this concept is to provide a much simpler intrinsic evolution [8]. The VRA concept is similar to the Cartesian Genetic Programming (CGP) [8] and provides other benefits, which includes (i) The VRA and the GA are housed in the same FPGA, making the communication faster (ii) The VRA is modeled in HDL making it easier to modify and synthesize in other FPGA target platforms (iii) The VRA architecture modeled can be utilized for similar problem definitions. Figure-8 shows the VRA structure implemented for the purpose. An array of configurable 'cells' is arranged in rows and columns. Each 'cell' input is connected to the outputs of the two previous columns with the exception of the first array column, which is connected to the inputs and its invert [9]. The architecture of the 'cell' is modeled based on the problem definition. Configuration bits from the GA provide the connectivity and logic based on the input combinations to the 'cell'.

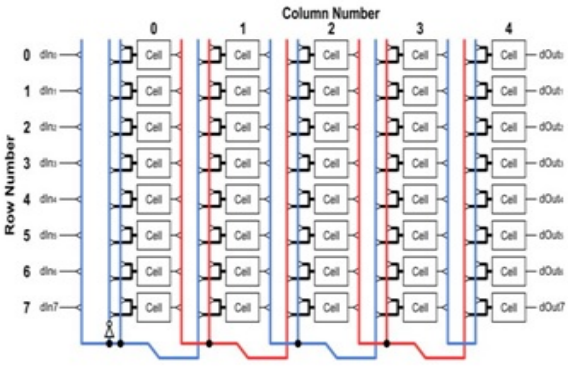

Figure-8. An 8 X 5 VRA core.

The evolvable embedded architecture is carried out on a ML605 Evaluation board with Virtex 6 FPGA. [15]. The use of Virtex 6 FPGA was mandatory due to simple and flexible implementation of a 32 bit soft core MicroBlaze processor [16]. This processor computes the GA, so faster evaluation time is achieved and also the complete EHW process can be implemented on a single chip.

4. SYSTEM DESIGN AND IMPLEMENTATION

This section explains the overall view of the complete evolvable embedded system design. The main components to be modeled are the configuration 'cells' of the VRA and the GA program for the optimization. The configuration 'cells' and the GA program is interlinked, as the logic and the interconnection of the cells in the VRA are based on the configuration bits of the GA program. The configuration 'cell' for the 2 bit adder circuit is modeled as shown in Fig 9. It consists of three 16:1 multiplexers, and an 8X1 bit RAM. Three multiplexers are used to select the inputs to the RAM (lookup table – LUT), which are driven from one of the set of sixteen inputs. A 20 bit configuration register is required to drive the 'cell' as illustrated in the figure, i.e., 3 * 4 select lines of MUX + 8 select lines to RAM. These configuration cells are interconnected in 'm' rows and 'n' columns to organize an array. For the design of a 2 bit adder an 8 x 5 cell array was selected, hence a total of 400 configuration bits (40 cells * 20 configuration bits) were asked to do the logical functions and interconnection between the cubicles. These configuration bits are provided by the GA program after matching the fitness at each iteration. The fitness measure is the truth table of the 2 bit adder circuit, having 5 inputs and 3 outputs. A fitness of 96 was calculated which has to match the hardware outputs from the cell array [8].

The evolutionary system was developed using Xilinx Platform Studio (XPS 14.6) tool. An embedded system with hardware and software elements was created in the EDK (Embedded Development Kit) and SDK (Software Development Kit) respectively; a 32 bit MicroBlaze soft core CPU with a clock frequency of 50 MHz was selected. The cell array VRA block was

imported as an IP core and interfaced as a peripheral to the Microblaze processor. Thus a hardware setup of the problem was synthesized using the EDK platform. A GA program has to be coded to configure the logic and interconnection, by matching the fitness criteria. Considering a large search space for finding the optimum solution, fitness matching was parallel performed for 8 array structures, to improve on the evaluation time.

A GA program with the above hardware setup was coded in the SDK. A simple GA program as coded using C language. The GA program was developed with the following functions:

(i) A simple GA program to provide a best fit configuration to the cell. The fitness function is the truth table of the 2 bit adder.

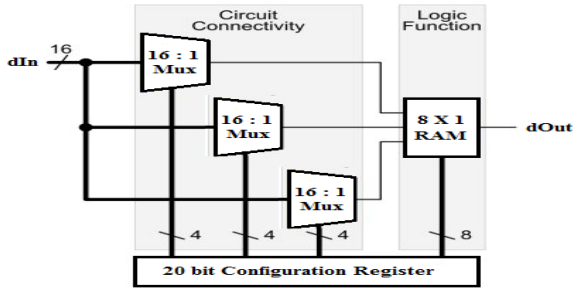

Figure-9. Architecture of the cell.

No. of iterations: 1000

Population size: 128 with 16 sets of population for each cell array (8 cell arrays in parallel)

Crossover rate: uniform crossover rate is used and tuned by trial and error method to get minimum generations.

Mutation rate: bit flip mutation is used and tuned by trial and error method to get minimum generations.

Selection method: tournament selection is implemented with the number of selections to be optimized based on the generations.

(ii) The truth table for a 2 bit adder circuit was loaded in the memory. The hardware fitness matching for 8 parallel arrays was performed by taking the values from the memory.

(iii) Configure the input and output registers of the array with the bit placing as shown in the Fig 10. For the 2 bit adder the first 5 bits of the input register represent the 2 bit input, and a carry (Cin, A0, A1, B0, B1) whereas the first 3 bits of output register represent the outputs (C0, S0, S1).

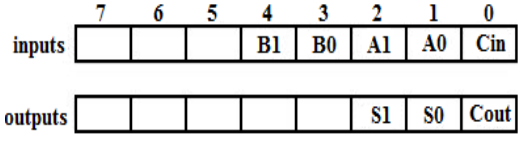

Figure-10. Input and output array configuration for a 2 bit adder circuit.

5. RESULTS

The results of the GA program are monitored on a PC. The results indicate the number of generations undergone to evolve a 2 bit adder circuit. The results on the PC indicate the number of generations to evaluate the fitness probability. The best and the optimum solution has to be displayed which would prove to evolve an optimal or novel circuit for a 2 bit adder circuit compared to our manual design method. The result would display the configuration bits generated to perform the logic and interconnections in the VRA core. The optimization of the GA program to perform the evaluation and evolution of the circuit is based on the variation of the GA parameters like crossover rate, mutation rate or the number selection. A variation of the GA parameters manually to optimize the number of generations in the evolution process was considered a bad option, as it would be time consuming. An adaptive GA program to measure the mutation rate and crossover rate could be foreseen as an option, wherein the program manipulates the crossover and mutation rates adaptively, depending on fitness criteria and the number of populations.

Here an experimental analysis was taken to illustrate the variance of the GA operators with the number of generations required to optimize the development process. Figure-11 -13, illustrates this experimental process graphically. From the graphs it can be seen that improper selection of the GA parameters would sometimes destabilize the GA program to converge to a solution. Figure-11 gives the graphical representation of the crossover rate with the number of evolutions. From the graph it can be determined that a crossover rate of 45% gives an optimum evolution in 88 generations, but a variation in the crossover rate from 40% to 60% would vary the number of generations to 156 and 302 respectively as shown in Figure-11.

Similar findings were observed in Figure-12 and Figure-13. Figure-12 demonstrates the variation of the mutation rate with the number of generations. Here it can be observed that an optimum value of 88 generations was attributed to mutation rate of 10%. A mutation rate of 20% would evolve the 2 bit adder circuit in 943 generations. From Figure-13 a tournament selection of 5 individuals was taken to get an optimum value of 88 generations. In summary, for an optimum 88 number of evolutions, a crossover rate = 45%, mutation rate = 10% and the number of selections = 5.

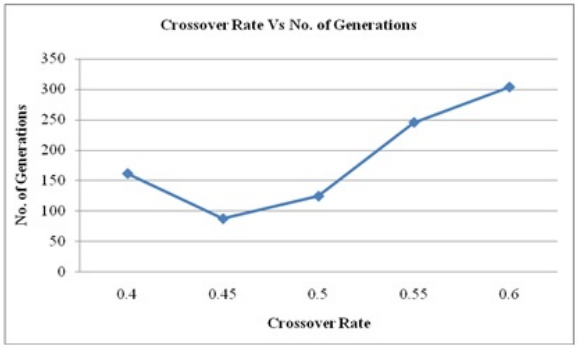

Figure-11. Variation of crossover rate vs. no. of generations.

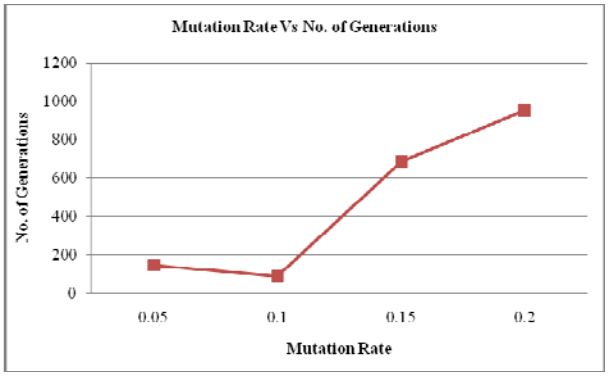

Figure-12. Variation of mution rate vs. no. of generations.

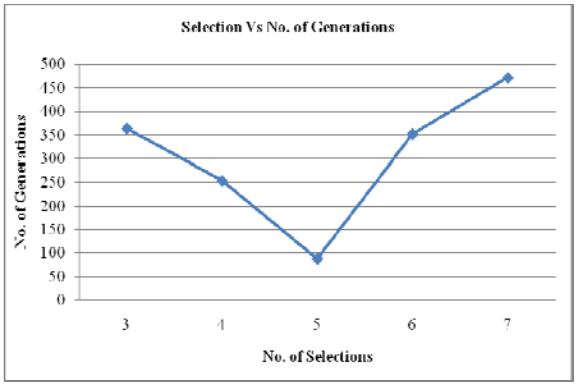

Figure-13. Variation of selections vs. no. of generations.

A complete evolution of a 2 bit adder using the above parameters was validated with the GA program. The results are shown in Figure-14. From the Figure it is validated that 88 generations were required to evolve a 2 bit adder circuit. The least number of generations was observed after varying the operators of the GA program.

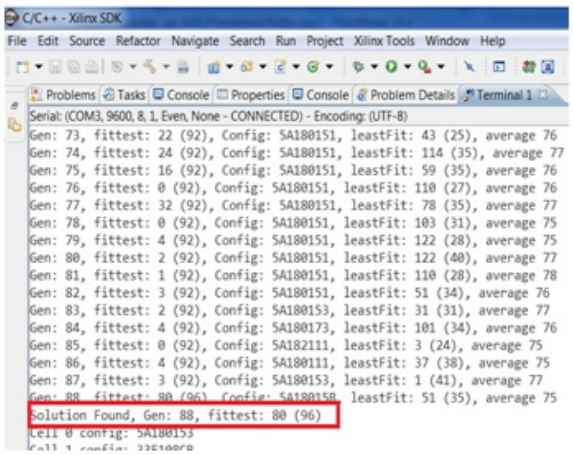

Figure-14. GA configuration generation for a 2 bit adder circuit.

5. CONCLUSIONS

This paper describes the importance of crossover, mutation and the number of selections in the working of a GA program. These GA operator values have to be tuned, such that optimum numbers of generations are required in obtaining a solution. Usually the crossover rates are maintained high, so the off springs attain the maximum fit chromosomes of the parent. Likewise the mutation rates are kept low, to prevent the GA from producing a large number of random solutions. A tournament selection mode is used to take the best fit chromosomes to the next generation. This process increases the speed of convergence for a GA. The above conditions are theoretically proved. In this experimental setup, it can be seen that a minimal number of generations were observed when the crossover rate was 45% rather than a 60% mutation rate. Likewise the same holds good for mutation rate and the number of chromosomes in tournament selection which are taken to be 10% and 5 respectively to get the minimum number of generations for our evolution. Therefore, this paper can be concluded that theoretical and practical analyses show high crossover rates and low mutation rates, but it also shows that, the GA operators are problem dependent. They attain optimum value for the minimum number of generations at certain levels only, and that the designer has to choose the values intelligently for a better GA convergence.

REFERENCES

[1] A E Eiben and J E Smith. 2003. "Introduction to Evolutionary Algorithm", Springer, Natural Computing Series, 1st Edition, ISBN 3-540-40184-9.

[2] Garisson W Greenwood and Andrew M Tyrell. 2007. "Introduction to Evolvable Hardware: A practical guide for designing Self Adaptive Systems", IEEE press series on Computational Intelligence. John Wiley Intercsience publication.

[3] Jim Torresen. 2004. “An Evolvable Hardware Tutorial”, Field Programmable Logic and Application, Lecture Notes in Computer Science Vol. 3203, , pp. 821-830.

[4] Lukas Sekanina and Vladimır Drabek. 2004. “Theory and Applications of Evolvable Embedded Systems”, Proceedings of the 11th IEEE Computer-Based Systems (ECBS’04), IEEE Computer Society Press, August.

[5] Vedavathi. A, Meena. K.V. and Gayatri Malhotra. 2012. "VHDL Implementation of Genetic Algorithm for 2-bit Adder", International Conference on Electronics and Communication Engineering, 20th, pp 57 – 63, May.

[6] Kumara Sastry, David Goldberg and Graham Kendall. 2003. "Genetic Algorithm", Springer, Introductory

Tutorial in Optimization and Decision Support Techniques, pp. 97 – 125.

[7] P. Soleimani, S. Mirzakuchaki, K. Mohammadi and M. Bagheri. 2011. "A Novel Evolutionary design of Sequential Logic Circuits by using Genetic Algorithm", International Journal of Modeling and Optimization, Vol. 1, No. 3, August, pp. 231 – 235.

[8] Lucas Sekanina and Stepan Friedl. 2008. "An Evolvable Combinational Unit for FPGAs", Computing and Informatics, Vol. 23, pp 461- 486.

[9] Andrew Greensted, "Evolving Digital Hardware", Lecture 1, EHW Module.

[10] Lucas Sekanina. 2006. "Evolutionary Design of Digital Circuits: Where are current limits?", Proceedings of the first NASA/ESA Conference on Adaptive Hardware and Systems, IEEE.

[11] Kaifeng Zhang, Huanzhang Lu, Weidong Hu and Jian Wang. 2014. "A LUT manipulation based intrinsic evolvable system", IEICIE Electronics Express, Vol. 11, No. 4, pp 1-7.

[12] J Wang, Q.S Chen and C. H. Lee. 2008. "Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware", IET computers and Digital Techniques, Vol. 2, No. 5, pp 386 - 400.

[13] Ruben Salvador, Andrea Otera, Javier Mora, Eduardo de la Torre, Teresa Riesgo and Lucas Sekanina. 2013. "Self Reconfigurable Evolving Hardware system for Adpative Image Processing", IEEE Transactions on Computers, Vol. 62, No. 8, Aug.

[14] JingXia Wang and Sin Ming Loo. 2010. "Case study of Finite Resource Optimization in FPGA using Genetic Algorithm", IJCA, Vol. 17, No. 2, June, pp. 95 - 101.

[15] Xilinx. 2012. "ML605 hardware User Guide", Application UG534, October.

[16] Rod Jesman, Fernando Martinez Vallina and Jafar Saniie. "MicroBlaze Tutorial Creating a Simple Embedded System and Adding Custom Peripherals Using Xilinx EDK Software Tools", Embedded Computing and Signal Processing Laboratory, Illinois Institute of Technology.