Implementation of Genetic Algorithm Using FPGA with Applications

Dr. Hanan A. R. Akkar * & Omar Arif Abdul-Rahman *

Received on: 14/1/2008
Accepted on: 10/6/2009

Abstract

This paper presents a design of a general purpose GA. It is described using the Very high speed integrated circuits Hardware Description Language (VHDL), which is one of the Hardware Description Languages (HDLs). The complete system is implemented in a single Field Programmable Gate Array (FPGA) platform using the Foundation 2.1i, which is a software tool from XILINX. The feasibility of the purposed FPGA-based GA (FPGAGA) is demonstrated by testing it using two case studies. The objective of the first test is to maximize mathematical functions ($f(x) = 2x$, $f(x) = x+5$ and $f(x) = 2x^3 - 45x^2 + 300x$) over the domain $0 \le x \le 15$. Simulation results show that the adopted design is able to find the maximum obtainable values for all the functions. The objective of the second test is to solve circuit partitioning problem by distributing given modules (or cells) over two blocks. The number of inter-block connections and the number of modules assigned for each block should be minimum. Simulation results show that the final arrangements reached by the FPGAGA for the given arrangements (consist of 5, 10 and 15 modules) is well optimized in term" of the number of inter-block connections and the modules assigned for each block.

تنفيذ الخوارزميات الجينية بأستخدام مصفوفة البوابات المبرمجة مع التطبيقات

الخلاصة

أن هذا البحث يقدم تصميم لخوارزمية جينية متعددة الأغراض. ثم يتم وصفها بأستخدام لغة وصف المكونات الصلبة للدوائرة المتكاملة الفائقة السرعة (VHDL) والتي تعد أحد لغات وصف المكونات الصلبة (HDLs). يتم تنفيذ النظام الكلي على أحدى رقائق مصفوفة البوابات المبرمجة (FPGA) وذلك بأستخدام البرنامج الأساس (i 2.1), وهي أداة برمجية توفرها XILINX (المصنع لرقيقة مصفوفة البوابات المبرمجة المنتقاة لهذا العمل). وللتأكد من كفاءةالخوارزميات الجينية المنفذة بأستخدام مصفوفة البوابات المبرمجة(FPGAGA) فقد تم اختبارها بأستخدام حالتين تطبيقيتين. الحالة التطبيقية الأولى تهدف الى أيجاد الرقم الأكبر لمدى معين من الأرقام $0 \le x \le 15$وذلك بأسخدام المعادلات الرياضية ($f(x) = 2x$, $f(x)= x+5$, $f(x) = 2x^3 - 45x^2 + 300x$) وقد أظهرت نتائج المحاكاة الحاسوبية امكانية (FPGAGA)على ايجاد اعلى القيم المتاحة للدو ال المستخدمة.أما الحالة الثانية فهي تهدف الى حل مشكلة تقسيم الدائرة (circuit partitioning). حيث يشترط إعادة توزيع عدد من الخلايا على وحدتين (أو مجموعتين) شريطة أن توزع بصورة متوازنة وان ترتبط فيما بينها بأقل عدد ممكن من التوصيلات (أو الأسلاك) العابرة للوحدتين. لقد أظهرت نتائج المحاكاة الحاسوبية ان (FPGAGA) قد تمكن من ايجاد التوزيع الأفضل وذلك فيما يتعلق بعدد الخلايا المخصصة لكل قسم والتوصيلات العابرة للوحدتين لعدد من الترتيبات المعطاة (المكونة من 5 و 10 و 15 خلية).

*Department of Electrical &Electronics Eng.University of Technology, Baghdad/Iraq

1. Introduction

Genetic algorithms (GAs) are robust search and optimization algorithms that mimic the theory of evolution and natural selection. GAs were originally developed by John Holland in 1975 and have been applied to many applications, including power system control, fault detection, control systems, signal processing and circuit partitioning problems [1, 2]. However, the software implementation of GAs when applied to increasingly complex problems and a large search spaces, can cause unacceptable delays. Hence, an alternative to this approach is to implement a GA in hardware. The combination of pipelining and parallelism in a hardware implementation of a GA increases the speed of the system when compared to its software counterpart. This allows the hardware implementation to be applied to various complex applications [3, 4].

Since most of the GA's operations are simple, a hardware implementation is feasible. Although this was not the case before the advent of Field Programmable Gate Arrays (FPGA), e.g. those from Xilinx (an FPGA vendor) that are programmed via a bit pattern are stored in a Static Random Access Memory (SRAM). This is because a general-purpose GA engine requires certain parts of its design, notably the function to be optimized, to be easily changed. Typically the function to be optimized is the only component that requires changing in a general-purpose GA. The other operations are generic and can be implemented in an Application Specific Integrated Circuit (ASIC) or other non-programmable medium if desired [5, 6].

This paper describes the FPGA-based GA (FPGAGA), which is a self-contained implementation of a hardware-based GA. Because of the re-programmability of the FPGA, the proposed design is a general-purpose GA engine which is useful in many applications where conventional software-based GA are too slow. It is described using the Very High Speed Integrated Circuits Hardware Description Language (VHDL) to facilitate the scaling. On the other hand, this VHDL code is synthesized and implemented using the software from Xilinx (foundation 2.1i).

2. Previous Works in the Hardware GAs

GAs have been widely applied to various fields since 1975. There have been many examples of their applications in many fields especially in the engineering fields. This section presents examples of the GAs implementations to some of these applications. Finally, it covers several attempts of the FPGA implementation of GAs.

Graham and Nelson (1995) submitted a paper that described the Splash2 Parallel Genetic Algorithm(SPGA), which is a hardware GA that searches for optimal solutions to symmetric Traveling Salesman Problems(TSPs). This family of problems involves finding the shortest path through a collection of ***n*** cities, visiting each city exactly once and returning to the starting city. Each processor in SPGA consists of four FPGA and associated external memories and was found to perform 6.8 to 10.6 times the speed of equivalent software on a state-of-the-art workstation. Multiple processor SPGA systems, which use up to eight processors, find good TSP solutions much more quickly than single processor and software-based implementations of the genetic algorithm. The four-processor island-parallel SPGA implementation out performed all other SPGA configurations tested. This is the first known hardware implementation of a GA for this problem[4].

Turton and Arslan (1995) submitted a paper that proposed a new hardware based

Parallel Genetic Algorithm (PGA) using order-based crossover rather than the conventional two-point crossover operator. Thus, the proposed PGA would be capable of optimizing a new category of real-time combinatorial problems. Disc Scheduling is an example of such type of problems. It was found that the hardware implementation of this order-based genetic algorithm is capable of scheduling disc-access requests in approximately 2 milliseconds and that the PGA is more effective than any of the other techniques for optimizing the average access time [5].

Depending on what is published by Stephen et al (1997) "A Hardware Engine for Genetic Algorithms" [6], this represents an important paper for implementation of a general-purpose genetic algorithm using FPGA.

However, the current work attempts the following:

1. To write a VHDL source code for a general-purpose genetic algorithm, while, it is omitted from the above mentioned paper.
2. To implement the design using single FPGA platform using Spartan-XL series rather than 5 FPGA platforms using XC4003 series.
3. To implement the shared memory internally rather than externally. This will add speed merit to the design.
4. To implement more complex functions (circuit partitioning) rather than implementing only simple mathematical functions.

3. The FPGA-Based GA System Design

Conceptually, the FPGA-based GA (FPGAGA) fits into a general computing environment into the following way [6]:

The *front end* of the FPGAGA system consists of a simple interface program running on a personal computer or workstation. Here, the Foundation 2.1i and related technology is chosen as the front end of the FPGAGA. The GA parameters are written into the front end, which programs them into the internal memory of FPGA. Additionally, the code that describes the GA behavior is written using a specific language (VHDL in this case) into the front end. Then the front end translates the specifications into a hardware image and programs the FPGA, which in turn implements the GA. The FPGAGA operates upon receiving a *(go)* signal from the front end. When the system detects a high *go* signal, the system operates based on the parameters which already programmed in the internal shared memory. Currently the only used termination condition is a fixed number of generations specified by the user. Therefore, the process of computation repeats until the termination criterion reached. At that time, the system sends a *done* signal to the front end. Hence, the final population will be ready to be viewed and analyzed or further processing can be performed.

4. The Architecture of FPGAGA

The FPGAGA (Figure (1)) is designed to satisfy several criteria. First, the components should be simple and easily scalable. This allow us to arbitrarily scale the design (e.g. allow for larger population members) as compatible with the FPGA platform's constraints that the FPGAGA is implemented on. Second, due to the modular design simplicity, some modules can exploit the concurrency within themselves. To this end, the SM (Selection Module) and the CMM (Crossover/Mutation Module) operate on two population members simultaneously. Finally, the design should exploit the advantages of pipelining. Note that in Figure (1) when a module completes a task it can

immediately awaits more input to repeat the processing. Due to pipelining, GA operations do not have to be suspended while other GA operations run. The modules in Figure (1) are patterned after the GA operators defined in Goldberg's simple genetic algorithm (SGA). The FPGAGA modules operate concurrently with each other and together forms a coarse grain pipeline. The basic functionality of the FPGAGA design of Figure (1) is as the following [6]:

a) The front end starts the operation of the FPGAGA by sending a *go* signal to the memory interface and control module MIC (Memory Interface and Control Unit). The MIC acts as the main control unit of the FPGAGA during the start-up and shut-down and it is the FPGAGA sole interface to the outside world. After start-up and before shut-down, control is distributed, all modules operate autonomously and asynchronously.
b) The MIC notifies the Fitness Module (FM), the Crossover / Mutation Module (CMM), the pseudo-Random Number Generator (RNG), and the Population Sequencer Module (PSM) that the FPGAGA begins execution. Each of these modules requests its required user-specified parameters from MIC, which reads it from the internal shared memory.
c) The PSM starts the pipeline by requesting population members from MIC and passing them along to the selection module SM.
d) The task of SM is to receive new members from PSM and judge them until a pair of sufficiently fit members is found. At that time, it passes the pair to the CMM, rests itself, and restarts the selection process.
e) When the CMM receives a selected pair of members from the SM, it decides weither to perform the crossover and mutation based on the values sent from the RNG. When done, the new members are sent to the FM for evaluation.
f) The FM evaluates the two new members from the CMM and writes the new members and their fitness's at the internal memory via the CM. The FM also maintains information about the current state of the FPGAGA that is used by the SM to select the new members and by the FM to determine when the FPGAGA is finished.
g) The above steps continue until FM determines that the current FPGAGA run is finished. It then notifies the MIC about the completion; MIC in turn shuts down the FPGAGA modules and sends a *done* signal to the front end.

5. FPGAGA Simulation, Implementation and Evaluation of Results

The FPGAGA system that has been implemented in this work consists of seven modules, the pseudo-Random Number Generator (RNG), the memory module, the Memory Interface and Control module (MIC), the Population Sequencer Module (PSM), the Selection Module (SM), the Crossover/Mutation Module (CMM) and the Fitness Module (FM). Here, Fitness module is used to maximize simple fitness functions, which are as the following:

1. $f(x) = 2x$.
2. $f(x) = x + 5$.
3. $f(x) = 2x^3 - 45x^2 + 300x$.

Then, it is used to solve the circuit partitioning problem for different module arrangements consisting of 5, 10 and 15 modules. In this paper all the above stated modules are described using the VHDL, except for the memory module which is created and added to the design using the Foundation CORE Generator (Foundation

design tool). Then, the Foundation synthesis and implementation tools are used to target the proposed design in the S40XLBG256, which is the chosen FPGA platform from Spartan-XL family, since it contains suitable hardware resources and employ distributed memory technology, which allows internal memory to be implemented in the design[9].

5.1 Steps of the Work

In this paper, the implementation of the FPGAGA design to the chosen platform involves the following steps;

a) The FPGAGA low level modules mentioned in the pervious section are described using the VHDL. The syntax of the module is checked and any error is removed using the Foundation HDL Editor (Foundation entry tool). Then Foundation Simulator (Foundation simulation tool) is used to perform the functional simulation to check the correct functionality of each module separately.
b) The user-specified parameters are written along with the initial population into a special format file. This file is used by the Foundation CORE Generator to load them in the internal memory. While the design's hardware parameters are written in the top level module, which is used also to connect the lower level modules.
c) The design is synthesized and implemented using the synthesis and implementation tools of the **Foundation 2.1i**. Then timing simulation is performed to check the performance of the implemented design. Simulation results are analyzed and the implementation reports are summarized.

5.2 The Hardware Parameters

The hardware parameters are defined in the ROOT Module. They provide the required scalability for the proposed design. The primary hardware parameters are as the following:

a) The width in bits of the crossover and mutation probabilities and the random numbers sent from RNG to CMM is denoted as **p**.
b) The maximum width in bits of the population members is denoted **n**.
c) The maximum width in bits of the fitness values is denoted **f**.
d) The precision used in scaling down the sum of fitness's for selection in SM is denoted **r**.
e) The size of the cellular automaton in RNG is denoted **casize**.
f) The maximum of the population is denoted **m**.
g) The maximum number of generations is denoted **maxnumgens**.

The other parameters defined in the ROOT Module depend entirely on the above parameters, which are as the following:

a) The width in bits of the random numbers sent from RNG to CMM is donated *logn*, which is the base-2 logarithm of *n*.
b) The maximum width in bits of the coded indices, which are sent by different modules to the MIC, is denoted *lognumparam*, which is the base-2 logarithm of the number of parameters specified in s.
c) The maximum width in bits of the coded indices, which are sent by the PSM to the MIC, is denoted *logm*, which is the base-2 logarithm of the *m*.
d) The size of FM's register used to store the maximum number of generations for a given application is denoted *logmaxng*, which is the base-2 logarithm of the *maxnumgens.*
e) The maximum width in bits of an item (a population member along with its fitness) that is stored in a memory

location is denoted *membw*, which is equal to the product of *f* plus *n*.

f) The maximum width in bits of user-specified parameters that is requested by FM is denoted *maxof_logmf_logmaxng*, which is equal to the maximum value between (*logm* + *f*) and *logmaxnumgens*.

g) The maximum width in bits of an item stored in the memory is denoted *maxof_memwidth*, which is equal to the maximum value among the *casize*, *p*, *membwidth*, *maxof_logmf_logmaxng*.

h) The width in bits of the address bus of the RAM is denoted *addrw*.

i) The width in bits of I/O data bus of the RAM is denoted *valw*.

j) The width in bits for the difference between *logn* and *n* is denoted *d*. This parameter is used by CMM to ensure that the crossover and/or the mutation operators respect the bit groups when produced new members.

The value for these parameters are determined as required by a given application and must be compatible with hardware constraints for the chosen FPGA platform.

5.3 Case Studies

To demonstrate the FPGAGA feasibility, the FPGAGA is first tested on simple fitness functions. Then it is tested on more complex fitness functions. At each test, the FM is modified to describe the fitness function of the given problem. Then a random initial population is created and evaluated (using the RNG and FM individually). Once the user-controlled and hardware parameters are specified, the Foundation synthesis and implementation tools are used to target the FPGAGA system in the given FPGA platform. This section presents the simulation results, the performance analysis and summarization of the implementation reports of these tests.

5.3.1 Simple Fitness Functions

The FPGAGA is tested by optimizing the functions f(x) =2x, f(x) =x+5 and $f(x) = 2x^3 - 45x^2 + 300x$ [6]. At each new test, the VHDL code of the state **Evaluation** of the FM is modified to describe the given function. The functional simulation is performed successfully. The correct functionality of the module is verified by scrutinizing the simulation results.

5.3.1.1 Simulation Results of the FPGAGA System Maximizing the Function f(x) = 2x

The parameters used to implement the FPGAGA in order to maximize the given function for this test (f(x) =2x) are shown in tables (1) and (2) (test 1). While the results of simulation that was performed after the successful implementation of the FPGAGA system are shown in Figure (2).
The maximum fitness value obtained measures the performance of the FPGAGA system. Also, the average fitness value measures how close the population members from the maximum value. The function is maximized over the domain $0 \le x \le 15$; hence the maximum obtainable fitness will be 30. It is obvious from the graph of Figure(2) that the FPGAGA system reached this value from the 2^{nd} generation (run), and maintains this value through the successive generations. Also, the average fitness increases gradually until it reaches the value of 30 at 11^{th} generation and successive generations. This indicates that all population members reached the value that achieves the maximum fitness from the 11^{th} generation. These results shows how good the FPGAGA is for maximizing the function f(x) = 2x. The entire simulation run took 12,673 clock cycles.

5.3.1.2 Simulation Results of the FPGAGA System Maximizing the Function f(x) = x + 5

The parameters used to implement the FPGAGA in order to maximize the given function for this test (f(x) = x +5) are shown in tables (1) and (2) (test 2). While the results of simulation that was performed after the successful implementation of the FPGAGA system are shown in Figure (3).

The function f(x) = x +5 is maximized over the domain $0 \le x \le 15$; hence the maximum obtainable fitness value will be 20. The simulation results obtained after 16 FPGAGA runs are shown in Figure(3). It is clear from the graph that maximum fitness obtained by the FPGAGA system reaches the value of 20 from the 2nd generation (run). While the average fitness increases gradually until it reaches the value of 20 at the 12th generation and successive generations. This indicates that the maximum fitness value obtained measures the performance of the FPGAGA system.

5.3.1.3 Simulation Results of the FPGAGA System Maximizing the Function f(x) =$2x^3$ - $45x^2$ +300x

The parameters used to implement the FPGAGA in order to maximize the given function for this test (f(x) =$2x^3$ -$45x^2$ +300x) are shown in tables (1) and (2) (test 3). While the results of simulation that is performed after the successful implementation of the FPGAGA system are shown in Figure (4).The function f(x) =$2x^3$ - $45x^2$ +300x is maximized over the domain $0 \le x \le 15$; hence the maximum obtainable fitness value will be 1125. The simulation results obtained after 16 FPGAGA runs are shown in Figure (4). It is clear from the graph that the maximum fitness obtained by the FPGAGA system reaches the value of 1125 from the 2nd generation (run) and the successive generations. While the average fitness increases gradually until it reaches the value of 1125 at the 7th generation and the successive generations. This indicates that all population members reach the value that gives the maximum obtainable fitness from generation 7th. These results show how good the implemented FPGAGA system is in maximizing the function f(x) =$2x^3$ -$45x^2$ +300x. The entire simulation took 12,993 clock cycles. Therefore, the difference is negligible if compared with the number of clock cycles of the previous tests simulation run. This is due to the fact that state **Evaluation** is executed in single clock cycle regardless of the complexity of the given function.

5.3.1.4 Summary of Implementation Reports

The Implementation reports can be summarized as follows:

a) The hardware cost for the simple fitness functions tests is summarized in table (3). It is clear that the hardware utilization is directly related to the complexity of the given application and the technique of implementation. For example, the hardwire utilization at the case f(x) =$2x^3$ -$45x^2$ +300x is increased by 73% compared to the case f(x) = x + 5, while it is increased by 74% compared to case f(x) = 2x. On the other hand, the difference between case f(x) = x + 5 and f(x) = 2x is negligible (about 1%).
b) In all tests, the design is implemented on the same platform to ease the comparisons. The hardware utilization is 55% for the case f(x) = 2x, and it is 56% for the case f(x) = x + 5, while it is 97% for the case f(x) =$2x^3$ - $45x^2$ +300x. Therefore, the chosen platform has a suitable capacity to implement the given functions.
c) The processing time evaluation is shown in table (4). Here, the expected execution

time of the real system is related to the complexity of the given application and the technique of implementation. The maximum value of the worst case connection delay is (18.592ns), while the maximum value of the clock frequency with which the system operates properly is (19.558MHz). On the other hand, the longest expected execution time is (634.779 μs). These figures give an impression about how fast the hardware implementation of the GA is.

d) No errors are declared in all reports.

5.3.2 Case Two (Circuit Partitioning)

Circuit partitioning is the task of dividing a circuit into smaller parts. The objective of the circuit partitioning is to divide a circuit into parts as the sizes so that of the components are within prescribed ranges and the complexity of connections between the components is minimized. As the size of present-day computers chip increases (i.e. chips contain more than ten millions transistors in sub-micron areas), the interconnection delay (connection between two transistors) becomes a dominant factor over the gate delay. The interconnection delay is associated with the numbers and length of the connected wire, therefore one of the main objectives of the VLSI design is to limit delay within the circuit and allow for a higher clock frequency. Circuit partitioning is an important approach in reducing wire-length, and increasing the speed of the overall design. Hence, the main objectives that may be satisfied by the desired partition are [7, 8]:

- Minimization of the number of cuts (inter-partition connections).
- Minimization in the number of modules (elements) assigned to each partition.

Since circuit partitioning is showed to be an NP(Non-deterministic Polynomial time) complete problem, the proposed design of the FPGAGA is used to solve the partitioning problem.There is a scheme that has been proposed to apply to GAs to minimize the number of cuts. A design is composed of *c* components and a particular partitioning (population member) is represented by *c*-bit string P (as shown in figure (5)), where the *i*th bit is 1 if and only if component *i* lies in **block B**. Accompanying this bit string is a set of *c*-bit strings N_j, one per inter-component net in the design, where the *i*th bit of N_j is 1 if and only if component *i* is connected to N_j. So net j lies in **block B** of partition P if one of the bits in $P \wedge N_j$ is 1, where $\wedge$ is the bitwise AND operator. Likewise, net *j* lies in **block A** of partition P if one of the bits in $\overline{P} \wedge N_j$ is 1, where $\overline{P}$ is the bitwise NOT of the P. Thus, a net *j* crosses a chip boundary if and only if some bits from $P \wedge N_j$ is 1 and some bit from $\overline{P} \wedge N_j$ is 1. This can easily be determined with combinational logic. A partition cost fitness (F_{cut}) will be, then, the total number of boundary crossing. Since our objective is to minimize the cost fitness, the following cost-to-fitness transformation:-

$$F_P = F_{max} - F_{cut} \quad \ldots (1)$$

Where F_{max} is an input coefficient and it will be chosen as the maximum number of inter-component net for the initial circuit arrangement [6].

Then a control will be used to check weither the given partition respects the prescribed partition size (the number of modules assigned to each partition should not exceed 40%-50% of the total number of modules). Any partition violating this range will be punished by setting its resultant fitness (F_R) to 1. Consequently, its chances of survival in the next generations will be minimized.

This approach will be implemented in the following manner (as shown in figure(6)). First store P in the register of length *c*-bit. Each bit represents which of the block **A** or **B** its components lies in. then for each net N_j, a counter initializes itself to 0 and cycles through the values $v \in [0,1]$. For each value *v*, compare it to P_i (the index of the block holding component *i*) for all *i*. If they are equal, then component *i* lies in block *v*. The results for all i are logically ORed yielding a single bit indicating which partition that N_j lies in. This result is fed into an accumulator which counts the number of blocks that N_j lies in. After looping through all values $v \in [0,1]$, the accumulator is checked. If it holds a value equal to 0, then N_j crosses a chip boundary and F_{cut} will be updated. This process will be repeated for all N_j. Finally, F_P will be evaluated as defined by Eq.(1). Then the number of zeros in the given partition will be counted. If the resultant number is within the predefined partition size, F_R will be set to F_P. Otherwise, F_R will be set to 1.

In this paper, The FPGAGA is used to solve the partitioning for three different arrangements, which are of 5, 10 and 15 modules [8]. At each new test, the VHDL code of the state **Evaluation** of the FM is modified to describe the given arrangements. The functional simulation is performed successfully. The correct functionality of the module is verified by scrutinizing the simulation results.

The implementation and synthesis steps are performed successfully for all tests. This section presents the simulation results, performance analysis and summarization of the implementation reports for these tests.

5.3.2.1 Simulation Results of the FPGAGA System Optimizing the 5 Modules Arrangement

The parameters used to implement the FPGAGA in order to maximize the given function for this test (5-module arrangements) are shown in tables (5) and (6) (test 1). While the results of simulation that was performed after the successful implementation of the FPGAGA system are shown in Figure (7).

The simulation results, which is obtained after 16 FPGAGA runs, are summarized in the Figure (7). It is clear that the maximum fitness of 3 is obtained from the 2nd generations, while the average fitness increases gradually until it reaches the value of 3 at the 7th generation and the successive generations. This indicates that all population members reach the value that gives the optimum fitness of 3. By comparing the initial arrangement and the final arrangement reached by the FPGAGA in figure (8), it can be noticed how efficient the system is at reducing the inter-component connections and distributing the modules fairly over the two modules. The entire simulation run takes 21,305 clock cycles.

5.3.2.2 Simulation Results of the FPGAGA System Optimizing the 10 Modules Arrangement

The parameters used to implement the FPGAGA in order to maximize the given function for this test (10-module arrangements) are shown in tables (5) and (6) (test 2).

The simulation results, which are obtained from the FPGAGA simulation runs are summarized in Figure (9). It can be noted that an optimum fitness of 4 is obtained from the 2nd generation, while the average fitness reaches the value of 4 at the 14th generation. The initial and final arrangement reached by the FPGAGA are

shown in Figure(10). Here the efficiency of the FPGAGA is evident since the cut-set is reduced from 6 to 2 and the modules are distributed equally over the two blocks. The entire simulation run took 29,258 clock cycles. This figure represents an increase by 37% compared to the previous test number of clock cycles.

5.3.2.3 Simulation Results of the FPGAGA System Optimizing the 15 Modules Arrangement

The parameters used to implement the FPGAGA in order to maximize the given function for this test (15-module arrangements) are shown in tables (5) and (6) (test 3). While the results of the simulation that is performed after the successful implementation of the FPGAGA system are shown in Figure (11). It can be shown that an optimum fitness of 7 appears in the 4^{th}-9^{th} generations, and 13^{th} -16^{th} generations. On the other hand, the maximum average fitness value is 5.9375 at the 14^{th} generation. This oscillation in the maximum and average fitness is mainly due to the fact that the search space increases as the length of the chromosome increased. However, it is obvious that the final arrangement, which is shown in Figure (12), is well optimized in terms of the number of inter-blocks connections and the size of each block. The entire simulation took 41,865 clock cycles. Although the number of blocks is tripled compared with the first case the simulation time is increased by 97%. This increase in executions time clock cycles is mainly due to the execution of some parts of fitness function depending on the length of the given arrangement, however, the increase in the rate of execution time in term of clock cycles remain less than that of the number of modules for the given arrangement.

5.3.2.4 Summary of the Implementation Reports

The Implementation reports can be summarized as follows:

a) The hardware cost for the circuit partitioning problem tests is summarized in table (7). It is clear from the table that the hardware utilization is directly related to the number of the modules in the given arrangements and the technique of implementation. For example, the hardware utilization for the 15-module test is increased by 48% compared to the 5-module test, while it is increased by 19% for 10-module test.
b) In all the tests the design was implemented on the same platform to ease the comparisons. The hardware utilization is 58% for the 5 module test; it is 72% for the 10-module test, while it is 86% for 15 modules test. Therefore the chosen platform has the suitable capacity to implement the given arrangements.
c) The processing time evaluation is shown in table (8). Here also, the technique of implementation and the number of the modules in the given arrangements is the decisive factor. The maximum worst case connection delay is (13.875 ns), while the maximum clock frequency with which the system operates properly is (31.595MHz). On the other hand, the longest expected execution is (1325.51μs). These values emphasize the speed of the FPGA implementation for the GA.
d) No errors are declared in all the reports

6. Conclusions

This paper presents the FPGAGA which is a design to implement the genetic

algorithms on FPGA platform. The design is described using the VHDL which allows parameterized modules to ease scalability. Each of the FPGAGA modules is verified to be functionally correct through numerous simulations. After the correct functionality of each module has been verified, the modules are connected to each other and the system is synthesized and implemented successfully using the Foundation 2.1i synthesis and implementation tools. The FPGAGA possesses the speed of hardware while retaining the flexibility of the software implementation due to the reprogram ability of FPGA. Therefore the system is a general-purpose GA engine which is useful in many applications where software based GA implementations is too slow.

Simulations results for the FPGAGA maximizing the simple functions are good. In all tests the systems are able to find the maximum obtainable value for the given functions. This demonstrates that FPGAGA is an efficient system for maximizing the given functions.

The simulation results for FPGAGA when used to solve the circuit partition problem show the high performance of the adopted design. In each test the final arrangement reached by FPGAGA effectively minimized the cut-set and the number of modules assigned to each partition.

The number of clock cycles increases in relation to the complexity of the given problems, however the expected execution time emphasizes the FPGA implementation speed advantage and demonstrates that the FPGAGA successfully combined the benefits of efficient GA with the benefits of the FPGA implementation. It can be noted from the implementation reports that the technique used to implement the fitness function for a given application affects partially the overall processing time. Therefore a reduction in the processing time can be achieved by modifying the used technique to implement the fitness function for any given application since these modifications respects the area of the available FPGA platform. Processing time can be also reduced by increasing the system clock frequency which mainly depends on the state of art in FPGA technology. This technology will surely advance more in the future.

(6)

References

[1]Wikipedia, the free encyclopedia, ***"Genetic Algorithm***", http://en.wikipedia.org/wiki/genetic_algorthim, 2006.

[2] Holland, J. H., "***Genetic algorithms in search optimization and machine learning***", Ann Arbor, the University of Michigan Press, USA, 1992.

[3] Graham, P. S., "***A description, analysis, and comparison of hardware and a software implementation of the splash genetic algorithm for optimizing Symmetric traveling salesman problems***", M.Sc. Thesis, Brigham Young University, USA, October 1996, http://citeseer.ist.psu.edu/481234.html.

[4] Graham, P. and Nelson, B. ***"A hardware genetic algorithm for the traveling salesman problem on splash2"***, proceedings of the 5th international workshop on Field-programmable logic and applications, UK, pp.352-361, 1995, http://citeseer.ist.psu.edu/graham95hardware.html.

[5] Turton, B. C. H., Arslan, T. "***A parallel genetic VLSI architecture for combinatorial real-time applications-disc scheduling***", genetic algorithms in engineering systems: innovations and applications 12-14 September 1995, Conference Publication No 414 © IEE,

UK, 1995,http://citeseer.ist.psu.edu/turton95parallel.html

[6] Stephen D. S., Sharad S., and Ashok S. "***A hardware engine for genetic algorithms***", Technical report UNL-CSE-97-001, University of Nebraska-Lincoln, July, USA, 4, 1997, http:// citeseer.ist.psu.edu/scott97hardware.

[7] Kwon, Y., Park, B. and Kyung, C. "***SCATOMi: scheduling driven circuit partitioning algorithm for multiple FPGAs using time-multiplexed, off-chip, multicasting interconnection architecture***", Proceedings of the 21st International Conference on Computer Design (ICCD'03) 1063-6404/03, USA, 2003, http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/iccd/2003/2025/00/2025toc.xml.

[8] Pomeranz, I. and Reddy, S. M. "***Fault location based on circuit partitioning***", Proceedings of the 1996 International Conference on Computer Design (ICCD'96) 1063-6404/96, Netherlanda, 1996, http://www.iccd-conference.org/proceedings/1996/75540242.pdf.

[9] O. Arif Abdual-Rahman, Implementation of genetic algorithms using FPGA, M.Sc. Thesis, University of Technology, Department of Electrical and Electronic Engineering, Baghdad, 2007.

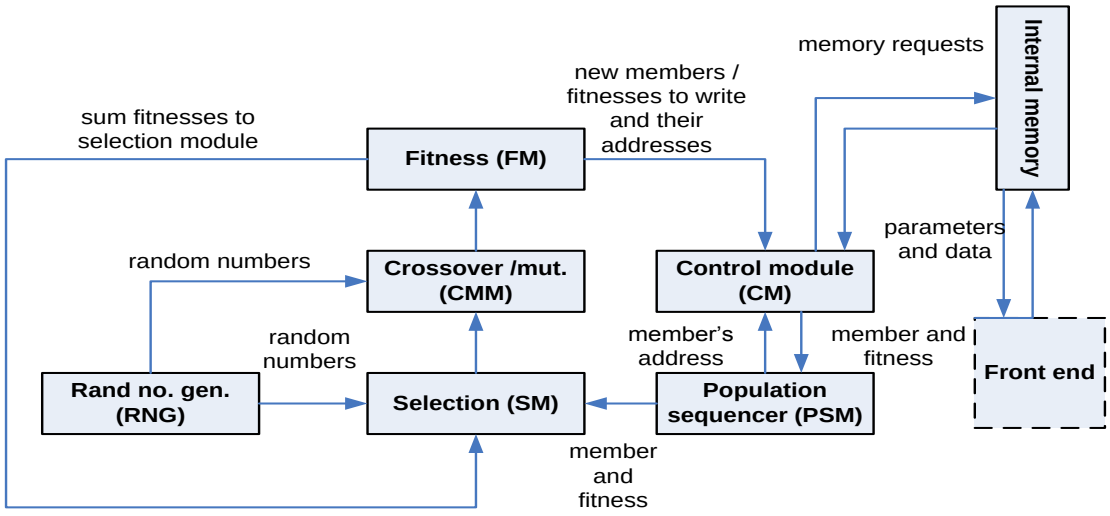

Figure (1) The architecture of the FPGAGA [6].

Table (1): User-Controlled Parameters for the simple functions test.

	Test 1	Test 2	Test 3
Size of population	16	16	16
The seed	"AAAA" (H^*)	"AAAA" (H^*)	"AAAA" (H^*)
The mutation probability	0.00195	0.00195	0.00195
The crossover probability	0.998	0.998	0.998
Initial sum of population	102	187	8965
Number of generations	16	16	16

***: refers to the value given in the hexadecimal system of numbers.**

Table (2): Hardware parameters for simple functions tests.

Primary parameters donation	Test 1	Test 2	Test 3
p	9	9	9
n	4	4	4
f	5	5	11
r	4	4	4
casize	16	16	16
m	16	16	16
maxnumgens	10	10	10

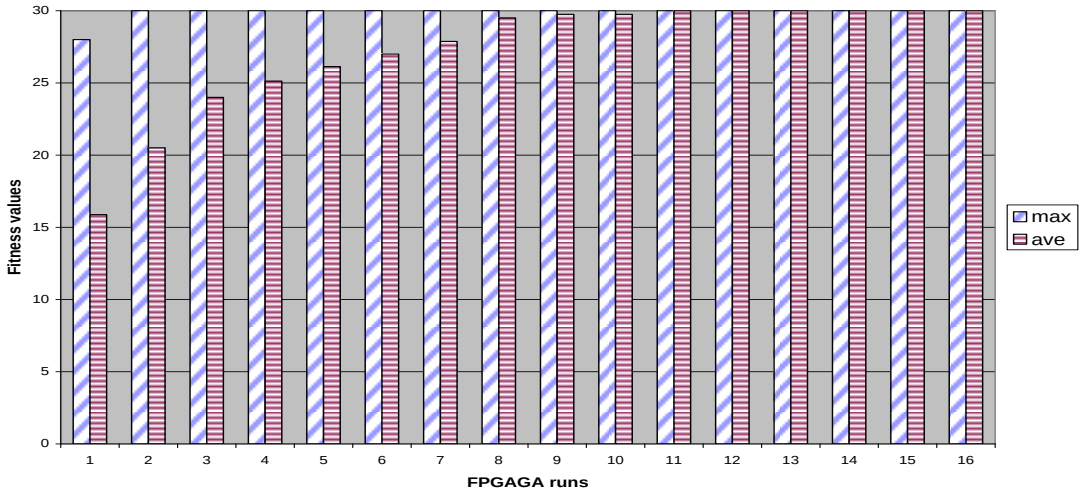

Figure (2) Simulation results for the case f(x) = 2x.

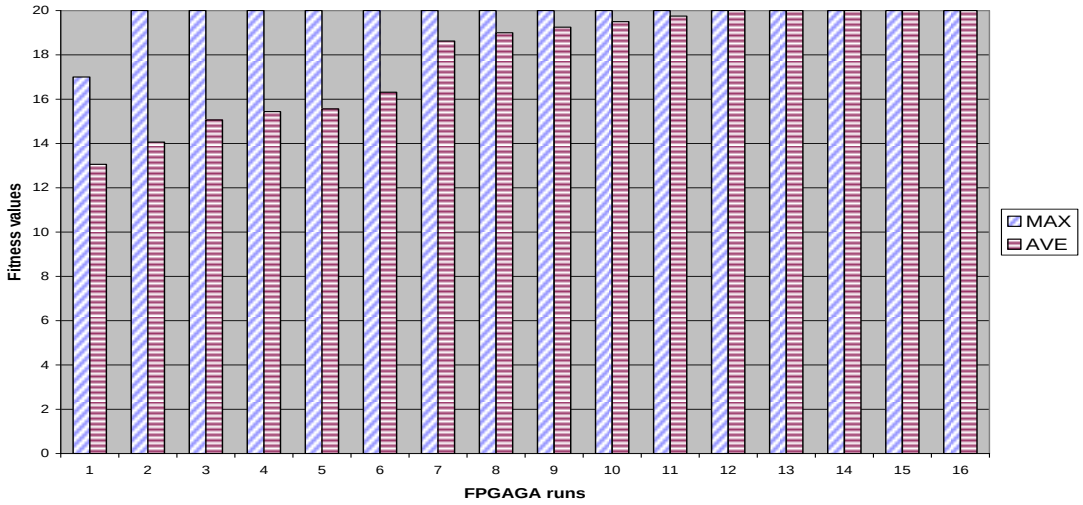

Figure (3) Simulation results for the case f(x) = x + 5.

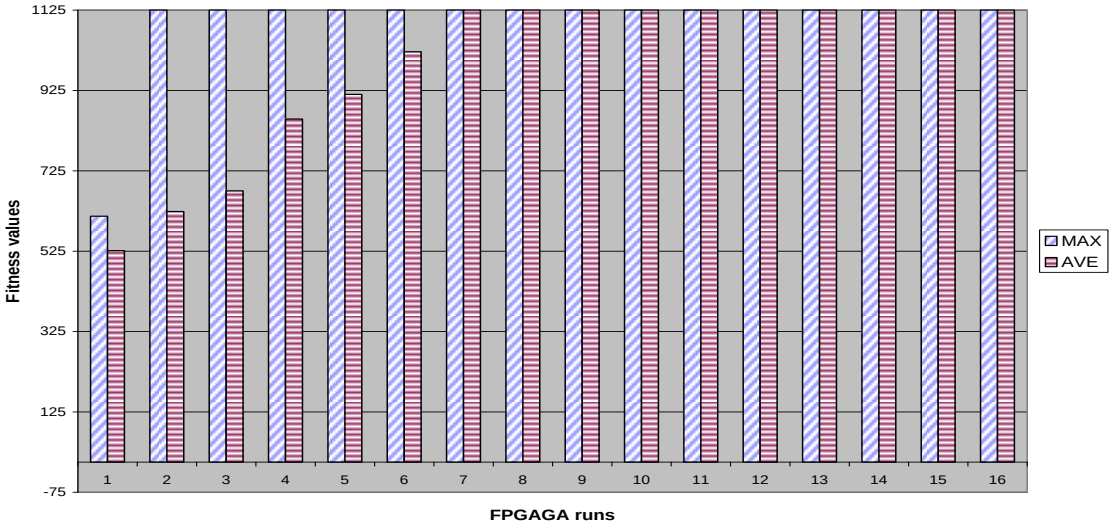

Figure (4) Simulation results for the case $2x^3 - 45x^2 + 300x$.

Table (3): Hardware utilization for the simple fitness functions.

	f(x) = 2x	f(x) = x + 5	$f(x) = 2x^3 - 45x^2 + 300x$
CLB	438	442	764
CLB F.F.	560	564	757
4input LUT	543	549	1013
3input LUT	149	149	195
16x1 RAM	72	72	120
I / O Blocks	36	36	36
Gate Count	12726	12730	21645
Add. Gates	1728	1728	1728

Table (4): Processing time evaluations for simple functions.

	f(x) = 2x	f(x) = x + 5	$f(x) = 2x^3 - 45x^2 + 300x$
Longest delay (ns)	12.892	14.812	18.592
Maximum frequency (MHz)	30.914	30.213	19.558
No. of clock cycles	12673	12889	12993
Expected execution time (µs)	409.944	426.604	634.779

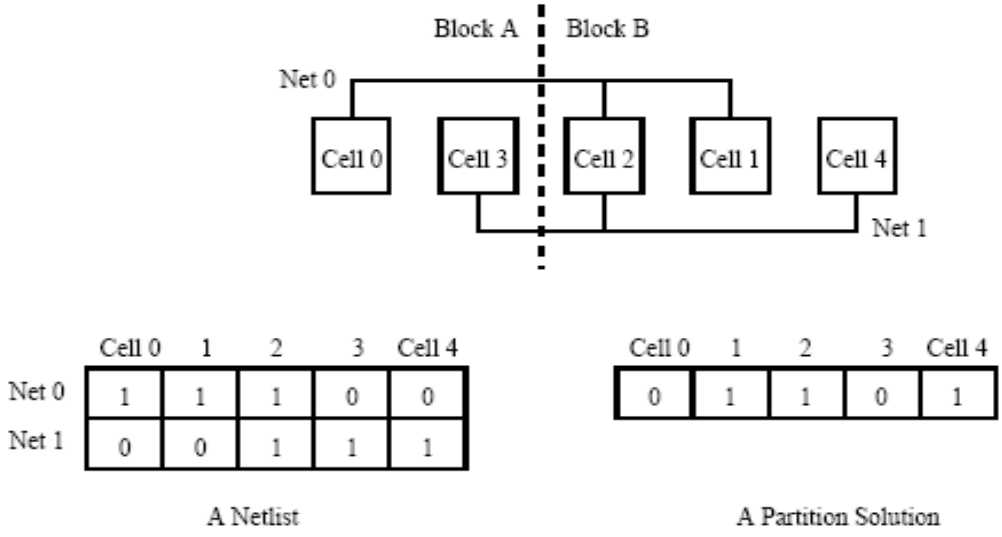

Figure (5) A Netlist and Partition Solution [8].

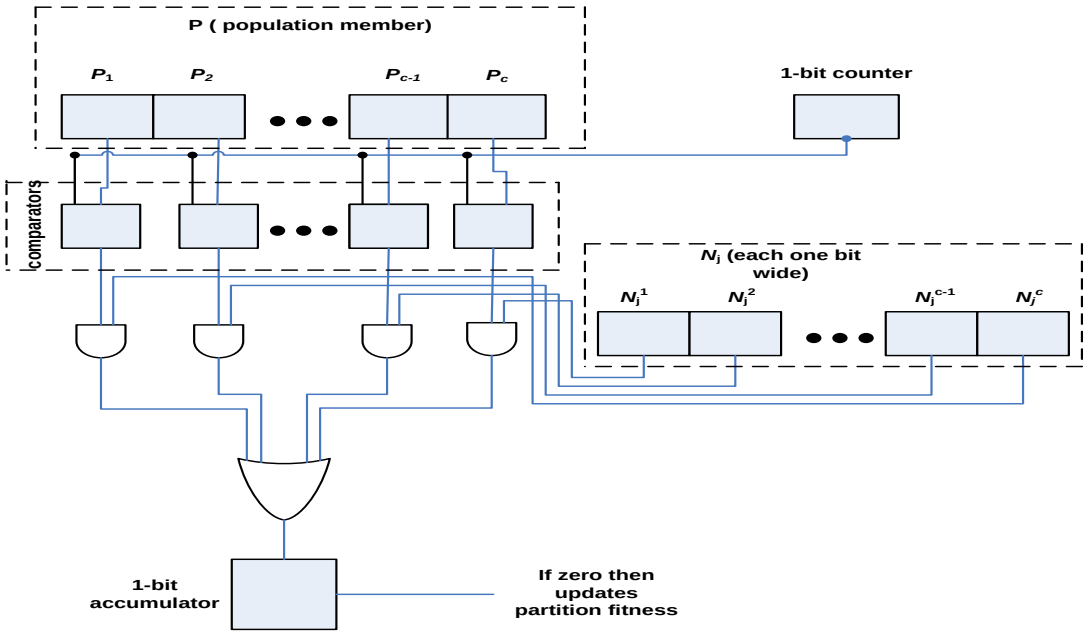

Figure (6) Circuit to evaluate 2-way partition.

Table (5): User-Controlled Parameters for the circuit partitioning tests.

	Test 1	**Test 2**	**Test 3**
Size of population	16	16	16
The seed	"AAAA" (H^*)	"AAAA" (H^*)	"AAAA" (H^*)
The mutation probability	0.00195	0.00195	0.00195
The crossover probability	0.998	0.998	0.998
Initial sum of population	19	22	28
Number of generations	16	16	16

*: refers to the value given in the hexadecimal system of numbers.

Table (6): Hardware parameters for circuit partitioning tests.

Primary parameters donation	**Test 1**	**Test 2**	**Test 3**
P	9	9	9
N	5	10	15
F	3	3	4
R	4	4	4
Casize	16	16	16
M	16	16	16
Maxnumgens	10	10	10

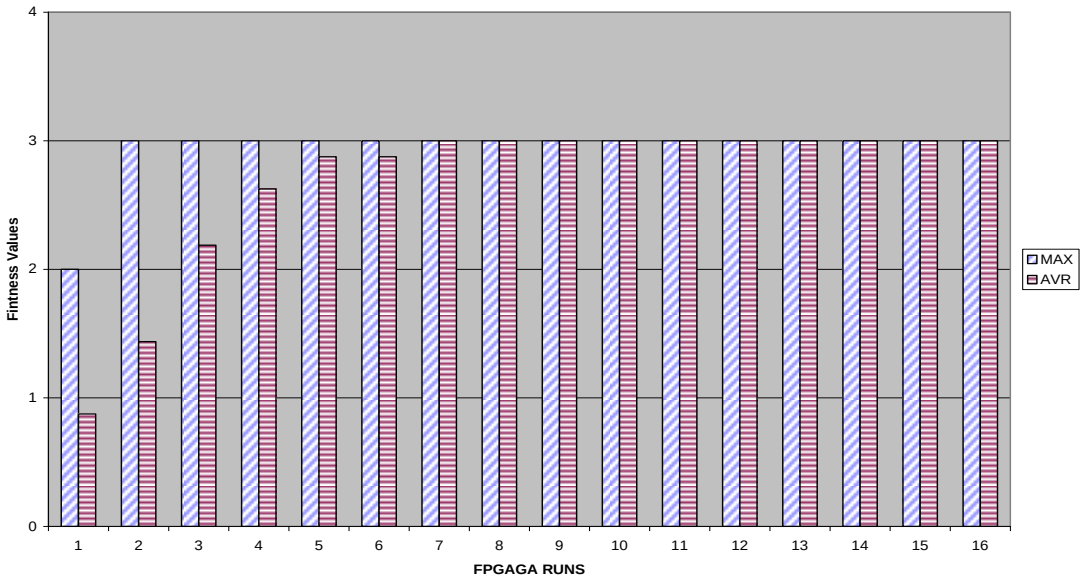

Figure (7) Simulation results for the 5-modules case.

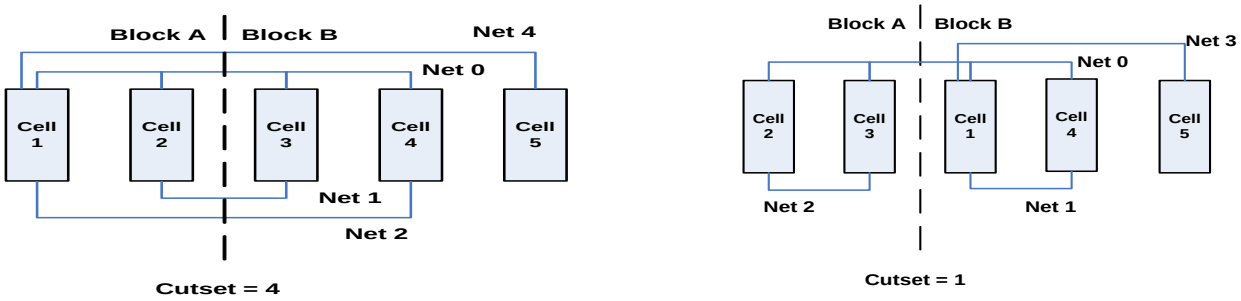

Figure (8) The initial and final arrangement that reach by the FPGAGA after 16 runs.

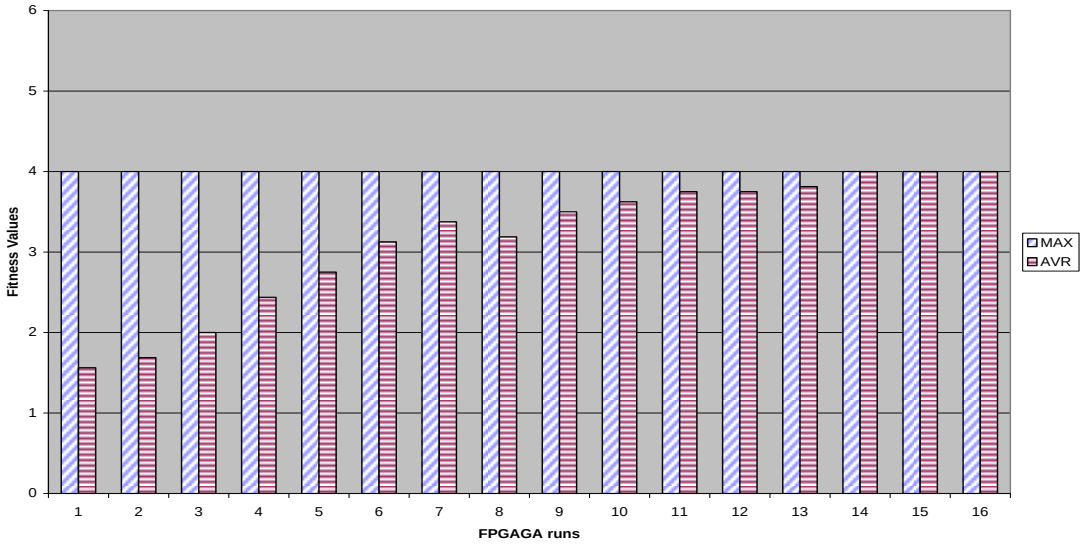

Figure (9) Simulation results for the 10-module case.

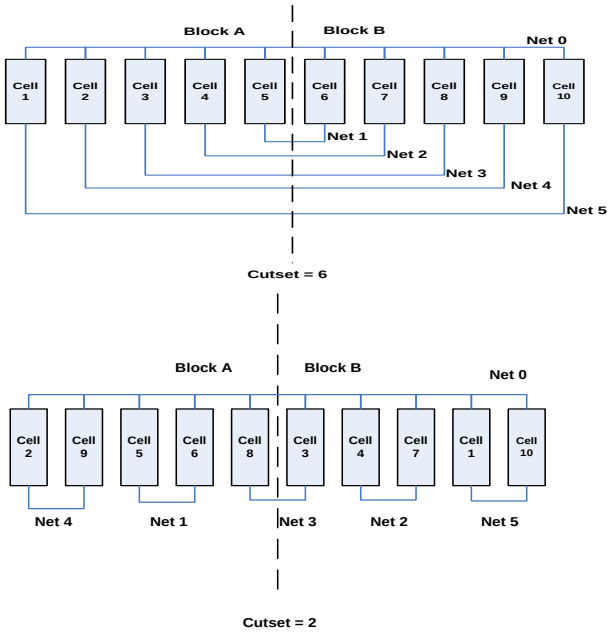

Figure (10) The initial and final arrangement that reach by the FPGAGA after 16 runs.

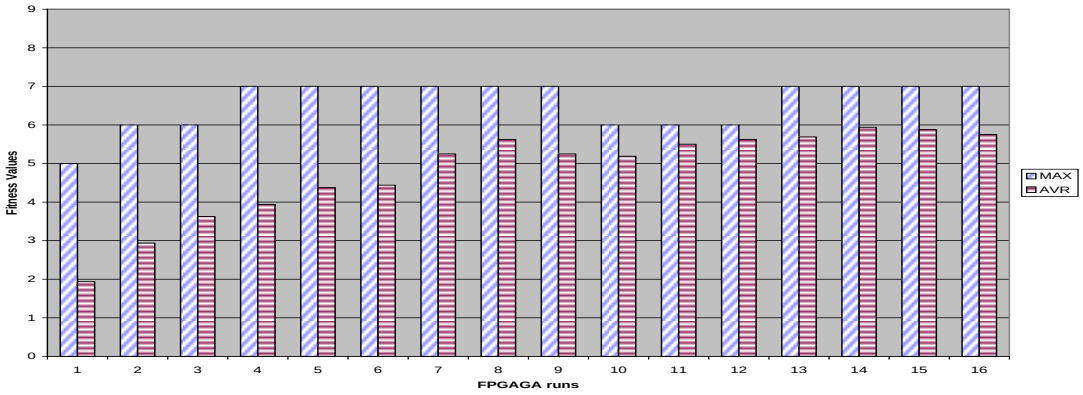

Figure (11) Simulation results for the 15-modules case.

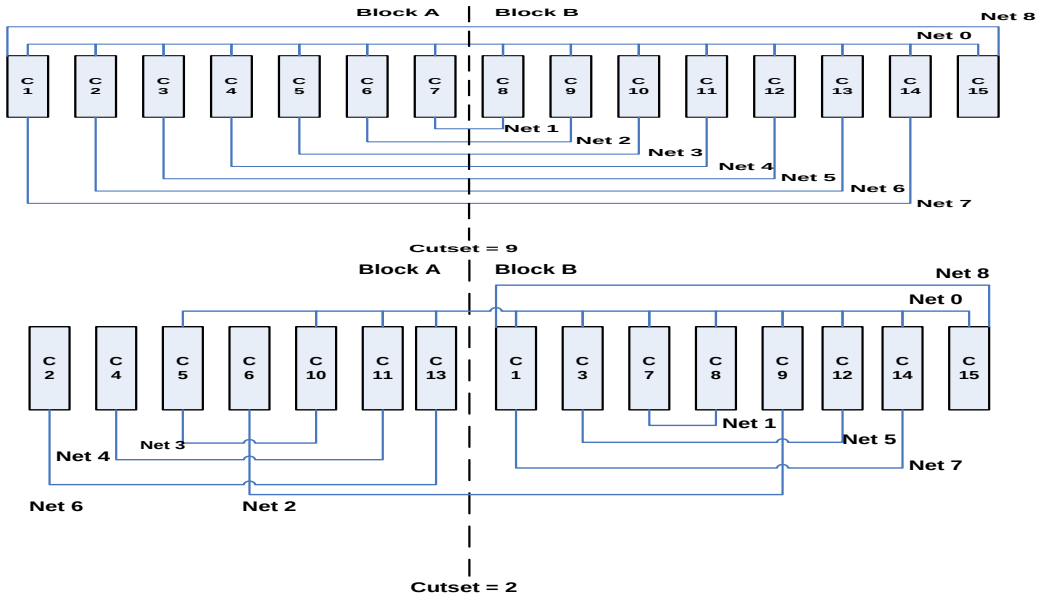

Figure (12) The initial and the best final arrangement that is reached by the FPGAGA after 16 runs.

Table (7): Hardware utilization for the circuit partitioning problem.

	5 modules	10 modules	15 modules
CLB	**459**	**566**	**678**
CLB F.F.	**593**	**759**	**948**
4input LUT	**596**	**707**	**800**
3input LUT	**153**	**188**	**239**
16x1 RAM	**64**	**104**	**152**
I / O Blocks	**36**	**36**	**36**
Gate Count	**12542**	**16890**	**21880**
Add. Gates	**1728**	**1728**	**1728**

Table (8): Processing time evaluations for the circuit partitioning problem.

	5 modules	10 modules	15 modules
Longest delay (ns)	11.300	12.407	13.875
Maximum frequency (MHz)	38.199	36.478	31.595
No. of clock cycles	21 305	29 258	41 865
Expected execution time (µs)	557.737	802.074	1325.51