TÜBİTAK

Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2016) 24: 5055 – 5067
© TÜBİTAK
doi:10.3906/elk-1502-122

Design and implementation of a genetic algorithm IP core on an FPGA for path planning of mobile robots

Adem TUNCER[1,*], **Mehmet YILDIRIM**[2]
[1]Department of Computer Engineering, Faculty of Engineering, Yalova University, Yalova, Turkey
[2]Department of Information Systems Engineering, Faculty of Engineering, Kocaeli University, Kocaeli, Turkey

Received: 16.02.2015 • **Accepted/Published Online:** 25.10.2015 • **Final Version:** 06.12.2016

Abstract: This paper presents a hardware realization of a genetic algorithm (GA) for the path planning problem of mobile robots on a field programmable gate array (FPGA). A customized GA intellectual property (IP) core was designed and implemented on an FPGA. A Xilinx xupv5-lx110t FPGA device was used as the hardware platform. The proposed GA IP core was applied to a Pioneer 3-DX mobile robot to confirm its path planning performance. For localization tasks, a camera mounted on the ceiling of the laboratory was utilized to receive images and allow the robot to determine its own location and the obstacles in the environment. In this way, procedures of path planning were tested in a real laboratory environment. An impressive time speedup was achieved when compared with its software implementation. Experimental results illustrate the effectiveness of the GA IP core hardware.

Key words: Genetic algorithm, path planning, mobile robot, field programmable gate array, intellectual property core

1. Introduction

Path planning is the most important task of autonomous mobile robots and it is a research area that is increasingly studied. Path planning is the process of finding an optimum collision-free path between a starting node and a target node in an environment with obstacles. Several methods and algorithms have been developed to overcome the path planning problems using various approaches, such as the grid-based A* algorithm, road maps (Voronoi diagrams and visibility graphs), cell decomposition, and artificial potential fields [1]. Comparing the methods with each other, each technique has its own advantages. However, they also have shortcomings depending on the type of application environment, such as a static or dynamic environment, and global or local planning. Recently, fuzzy logic, neural networks, and genetic algorithms (GAs) have been widely used to solve path planning problems.

The path planning problem can be addressed as an optimization problem when it is considered as the shortest or the least-cost path. The GA, which is a stochastic technique, is a robust search and optimization algorithm. It can find solutions for complex problems that traditional algorithms cannot find an optimal solution for in a reasonable time. It is able to reach a better solution quickly, since it searches all the search space simultaneously, in a parallel manner. The GA, which has become commonly used to solve optimization problems, has been utilized to find optimal paths in recent years [2–4].

However, when the complexity of the problem increases, the software implementation of the GA takes a long time to solve the problem, and this makes GAs unsuitable, especially for real-time applications. One

[*]Correspondence: adem.tuncer@yalova.edu.tr

solution to this problem can be the replacement of the software GA by a hardware equivalent [5,6]. Due to pipelining, parallelization, and the absence of function calls, a GA implemented in hardware generates a significant improvement in performance over a software GA [7]. This speed advantage makes hardware GAs very important for real-time applications. In this paper, it is intended to decrease the solution time or increase the speed by performing the GA as a hardware implementation.

The field programmable gate array (FPGA) has gained popularity in implementation hardware because of its low cost, fast design, and good performance. It is a digital integrated circuit that consists of programmable logic blocks, I/O interfaces, and interconnections between them. It is a general-purpose chip that is programmable and application-specific. It has a very wide range of applications. The FPGA is often used to accelerate applications that require high-speed computation by implementing them in hardware. This technology provides great flexibility to hardware designers in that that it can be reprogrammed unlimited times. One of the most important features of FPGAs is that they have the capability to perform parallel processing; therefore, they are frequently preferred in applications requiring high performance.

This paper shows how the path planning problem can be solved using a GA for mobile robots and a hardware implementation of the GA on an FPGA. The proposed study has several important features as given below; a GA is used for feasible and shortest path planning for mobile robots. A dynamic path planning, which was realized using a GA, was performed in our previous study [2]. The algorithm was tested in different environments and the success of the algorithm was demonstrated in that paper. However, in this study, we focus on the hardware implementation of the algorithm instead, to show the success of the algorithm.

In another of our previous studies [8], a hybrid implementation of a GA for path planning on an FPGA was carried out. Only the fitness function of the GA was implemented as a hardware intellectual property (IP) core while the other parts of the GA were implemented as software running over MicroBlaze, a soft microprocessor core from Xilinx. Speed differences between the IP hard core and the MicroBlaze implementations on the FPGA were examined. In that study, it was seen clearly that the speed of the IP core fitness function was increased. Because of this speed improvement, in this study, the overall GA design was implemented in hardware as a customized IP core on an FPGA using VHDL. The GA IP core was realized on a Xilinx xc5vlx110t FPGA device. The MicroBlaze soft processor on the FPGA was used as a communication unit between the GA IP core and the outside hardware. The coding was done using VHDL and simulated to test its correctness using Xilinx ISE 11.1. Some reported studies have used a hardware/software hybrid design, where some parts of the GA, especially the fitness function, are implemented in hardware [9,10] and the rest of the GA is implemented in software. However, in this study, all components of the GA are implemented in hardware. Digital image processing was used for the localization of the mobile robot, obstacles, and target. Within the frame of the study, the entire system was performed in a real laboratory environment using a camera and a Pioneer 3-DX mobile robot so as to demonstrate that the study is applicable in the real world as well as in simulation. In the literature, there are studies of FPGA implementations for general-purpose GAs. There are a small number of GA-based studies for path planning. Allaire et al. [11] proposed a solution for an unmanned aerial vehicle (UAV) with an autonomous real-time path planning capability based on a GA implemented on an FPGA. In that study, not all the operations of the GA were performed as hardware. The evaluation and mutation phases were not implemented on an FPGA. The mutation was only tested within a simulation environment. Kok et al. [12] proposed an autonomous GA-based UAV path planner with all modules of the proposed architecture implemented on an FPGA. Hachour [13] presented a hardware implementation of a navigation approach of an autonomous mobile robot on an FPGA. Unlike these studies, in our study all the components of the system were implemented as a customized IP core. Since there are no numerical results in the existing studies in the literature to make comparisons with, the software GA, MicroBlaze GA, and customized GA IP core were compared to each for a sample environment in this study.

2. Robot path planning with the genetic algorithm

As proposed by Holland [14], the GA is an optimization technique based on genetic science and has been applied to many hard optimization problems. It has been defined as one of the most powerful search algorithms for complex problems. Due to its parallel search capability, the GA can search the whole environment simultaneously. In recent years, the GA has been widely used to produce optimum paths by taking advantage of its strong optimization ability [4]. The GA is a method that outperforms others in path planning because of its capacity to explore the solution space while preserving the best solution that has already been found [11].

The GA requires the determination of the working environment of the robot and the coding of solutions into the chromosomes of the GA. Many path planning methods use a grid-based model to represent the environment [3]. The calculation of distances and the processing of obstacles are easier with grid-based representation [15]. Using the grid-based model, the environment can be viewed as a 2-dimensional coordinate plane.

In this paper, a 16 × 16 area is used for environment representation, as shown in Figure 1. A cell in the grid-based environment belongs to 1 of 2 categories that contain feasible or obstacle cells. In the figure,

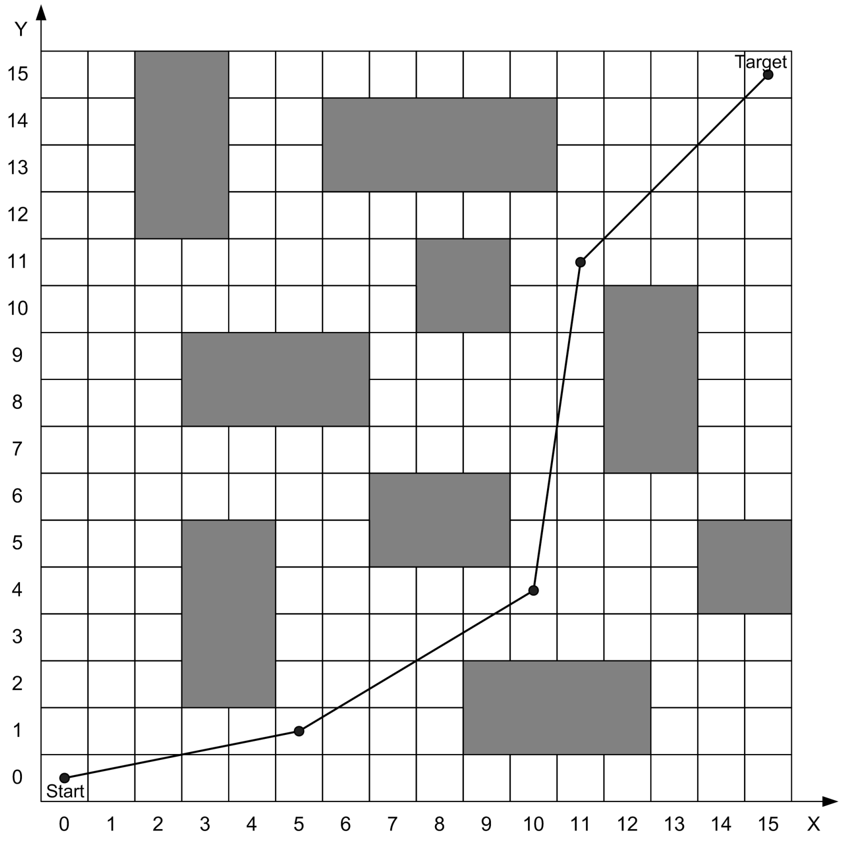


Figure 1. The 16 × 16 grid-based environment with obstacles.

a sample path from a starting node to a target node is also shown. A chromosome of the GA represents a candidate solution [16] for the path planning problem. A chromosome or a path consists of a starting node, a target node, and the nodes that the mobile robot goes over [2]. Each pair of genes in the chromosome contains the (x,y) coordinates for each point in the path. Figure 2 shows the chromosome for the coordinates of the sample path in Figure 1. As shown in the figure, a valid path consists of a sequence of grid labels, which begins from starting node and ends at the target node.

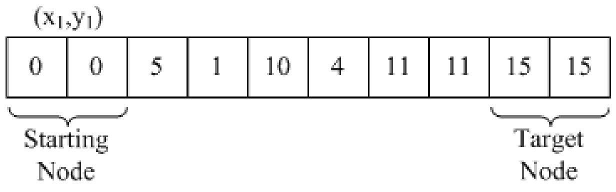


Figure 2. A sample path's coordinates.

3. Hardware implementation of the GA on an FPGA

In recent years, FPGAs have become very popular in embedded systems and high performance applications. FPGAs offer the advantages of hardware speed and software flexibility and hence are a good option for many scientific and engineering applications [10]. While designers can create effective hardware designs on FPGAs, many systems need both software and hardware together [17]. A whole system can be implemented as only software using C/C++, or only hardware using hardware description languages such as VHDL or Verilog. However, a part of the system can be implemented as software and the rest of it can be implemented as hardware. Many sophisticated FPGAs have soft processor cores that designers can use in their designs, such as Nios from Altera and MicroBlaze from Xilinx. Software cores are more portable and flexible than hardware cores. However, they consume more power, have lower performance, and require much more calculation time. In order to alleviate the performance and power overhead, a designer can potentially use hardware/software partitioning to increase software performance while decreasing energy consumption [17].

The soft processor in an FPGA provides wide flexibility for communication between software and hardware. The MicroBlaze softcore processor provided by Xilinx is highly configurable, allowing users to select a specific set of features required by the design [18]. This processor has 32-bit general purpose registers and a 32-bit address bus. The MicroBlaze instruction execution is pipelined.

In this paper, we propose putting a customized GA IP core into a MicroBlaze soft processor-based system. The more general view of the architectural structure of the FPGA-based GA is shown in Figure 3. The system consists of 6 basic modules, i.e. a random number generator module (RNGM), a fitness function module (FFM), a population sequencer module (PSM), a selection module (SM), a crossover module (CM), and a mutation module (MM). The initial population is generated by the RNGM. This population is then sent to the FFM, which evaluates the fitness value of each chromosome in the population. After that, the fitness values are sent to the PSM to sort chromosomes according to their fitness values. The SM selects parents for mating and sends them to the CM to cross the parents and generate offspring. Chromosomes are sent to the MM after the crossover and mutated there. The new population is sent to the FFM again. The RNGM runs in parallel with the other modules and supplies them with the required random numbers. The following briefly explains the hardware modules of the GA architecture.

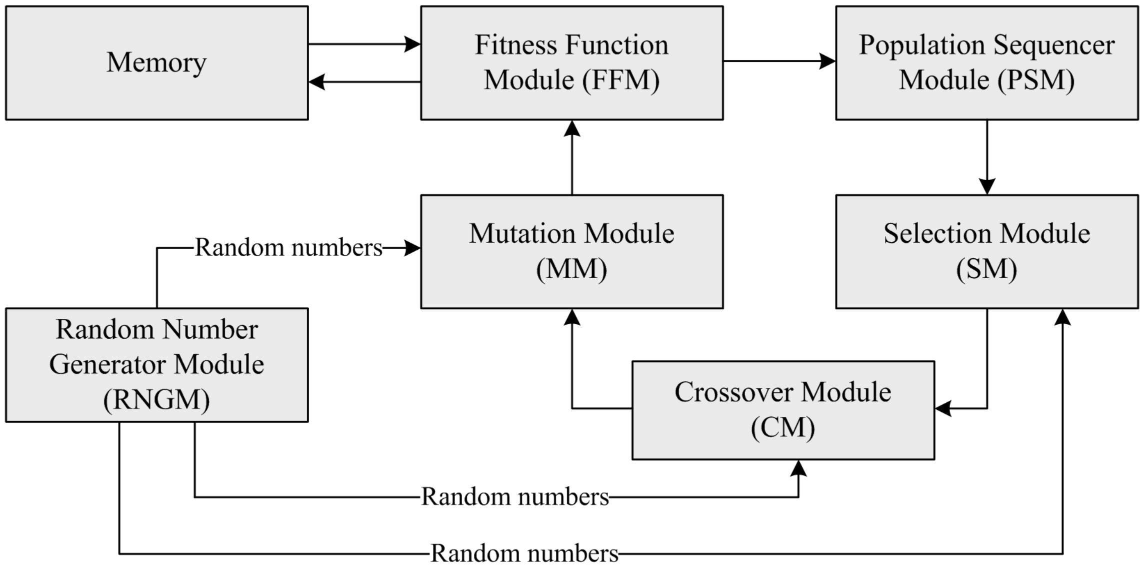


Figure 3. Architectural structure of the hardware GA on the FPGA.

3.1. Random number generator module (RNGM)

The RNGM is a very important module, which is required in the hardware GA. This module generates random numbers that are used for GA operators, such as the initial population, selection, crossover, and mutation. Generally, there are 2 methods for implementing the random number generator [19]: linear feedback shift register (LFSR) and linear cellular automata (LCA). However, LCA generates better random numbers than LFSR does [20]. In many studies, LCA has been used for the RNGM on FPGAs [19–21], and hence LCA is used in this study. The LCA consists of 16 cells, which are based on rule 90 and rule 150 as described by Wolfram [22,23] in Eqs. (1) and (2):

$$\text{Rule 90: } s_i^+ = s_{i-1} \oplus s_{i+1} \tag{1}$$

$$\text{Rule 150: } s_i^+ = s_{i-1} \oplus s_i \oplus s_{i+1} \tag{2}$$

Here, s_i is the current state of cell i, s_i^+ is the next state of state s_i, and $\oplus$ is the exclusive-OR operator. Serra [20] showed that a 16-cell LCA, whose cells were updated by the rule sequence 150-90-150, produces a maximum length cycle. Figure 4 shows a hybrid LCA that uses the structure of rules 90 and 150.

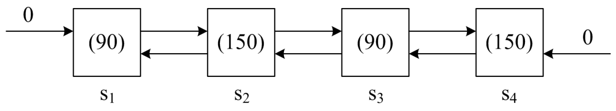


Figure 4. A random number generation sample with cellular automata [20].

In this study, 16-bit binary numbers are used for random number generation. Different bits of these binary number sequence are used for each GA operator. Figure 5 shows which bits of the 16-bit binary sequence are used for operators. Once the seed value is given, the RNGM efficiently generates a 16-bit random number on every system clock cycle of the FPGA.

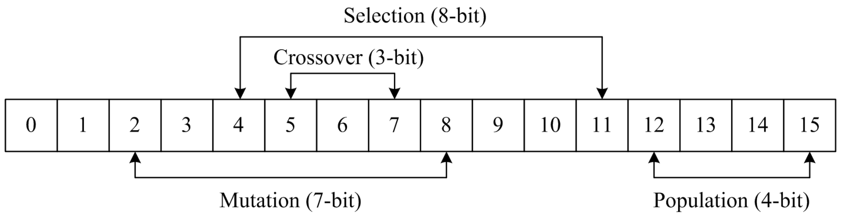


Figure 5. The distribution of a 16-bit binary number for operators.

3.2. Selection module (SM)

The main principle of the GA is that the best chromosomes should survive and be transferred to new generations. The selection process used for determining the best chromosomes in the population consists of 3 stages. In the first stage, the fitness function values of all the chromosomes are found. In the second stage, the fitness values are sorted. In the third stage, after selection according to fitness values, selected chromosomes are put into a mating pool to produce offspring chromosomes. The roulette-wheel selection method is used in the SM. In the proposed system, the roulette wheel selection method and elitism are used together. Elitism is provided by transferring the best chromosome in the current population into the next population. The SM receives an 8-bit random number from the RNGM.

3.3. Crossover module (CM)

This module crosses 2 selected parent chromosomes (paths) to generate 2 new offspring. A 3-bit binary random number supplied by the RNGM is used for the determination of a random crossover point. A single-point crossover method is used for the crossover process. The crossover point on the parent chromosomes is selected randomly for every cross. The offspring is generated by cutting at the chosen crossover point and swapping the second parts of the 2 parent chromosomes. Figure 6 shows a single-point crossover process.

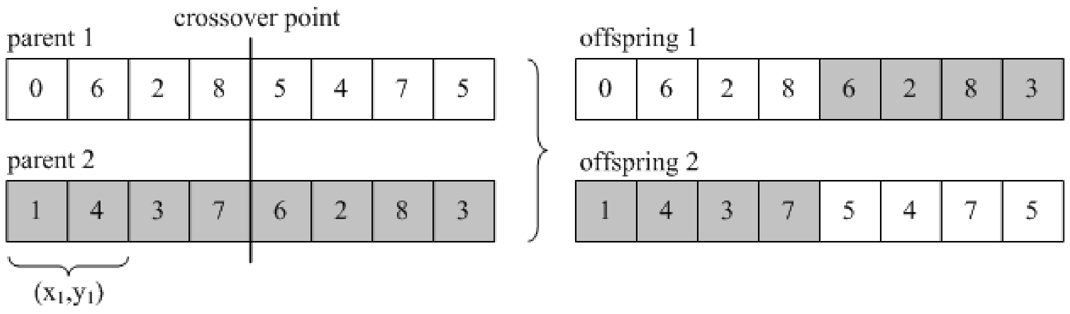


Figure 6. A single-point crossover.

3.4. Mutation module (MM)

The function of the MM is to randomly change the genes on the chromosomes in order to prevent them from resembling each other. After a new generation is created through the crossover operator, a new offspring is randomly chosen for the mutation process. How many genes in the population are changed is controlled by the mutation rate. The RNGM supplies random numbers for every gene and the MM compares the random numbers with the predefined mutation rate to decide whether mutation should be done for that gene. If a random number is smaller than the mutation rate, a new 4-bit random coordinate is generated for the mutated gene.

3.5. Fitness function module (FFM)

The fitness function is the sole problem-dependent part of a GA and it is formed according to the problem. The purpose of the path planning problem is to find a feasible (collision-free) path between a start point and a target point. The meaning of fitness herein may be the shortest path, the least time, or the least energy consumption. In this paper, the fitness function is considered as the shortest path and it is the Manhattan distance between the starting and target nodes. Bresenham's line algorithm [24] is used to determine the cells that construct the path. Unlike other algorithms, it uses only integer values instead of floating-point values, and bit shifting instead of division and multiplication, all of which are practical and quick. In this study, the fitness function value of a chromosome is calculated with Eqs. (3) and (4).

$$f = \begin{cases} \sum\limits_{i=1}^{n-1} d(p_i, p_{i+1}), \text{ for feasible paths} \\ \sum\limits_{i=1}^{n-1} d(p_i, p_{i+1}) + \sum\limits_{j=1}^{m} \text{ penalty, for infeasible paths} \end{cases} \tag{3}$$

$$d(p_i, p_{i+1}) = |x_{i+1} - x_i\ | + |y_{i+1} - y_i| \tag{4}$$

Here, f is the fitness function, p_i is the ith gene (node) of the chromosome, n is the length of the chromosome, m is the number of obstacles between 2 nodes, and d is the Manhattan distance between 2 nodes. As seen in Eq. (3), the fitness function is calculated as the sum of the distances between the genes of a chromosome. Penalty values are added to the fitness function if there is any obstacle between the 2 nodes. Thus, the selection probability of a path that passes through an obstacle is reduced by this penalty mechanism.

3.6. A customized GA IP core for path planning

IP cores are built-in functions that contain complex electronic circuit blocks. Many vendors offer IP cores to customers. The need for third-party IP cores in FPGA designs is driven by the increased size and performance of FPGA devices. Designers of these complex FPGA designs require proven IP cores to accelerate system development [25]. Although vendors provide IP cores, users can develop their own IP cores according to their problems and designs. In this paper, we designed a customized GA IP core for path planning of a mobile robot on an FPGA. The proposed GA IP core was developed by using VHDL, including initial population, fitness, population sequencer, selection, crossover, mutation, and random number generator modules.

The overall GA was designed and executed as a hardware IP core on an FPGA. The connection between the FPGA and the outside environment was carried out via the MicroBlaze soft processor. These connections include the camera and robot connections. The map information is sent to the GA IP core on the FPGA from a computer, which is connected to a camera. The fitness function is calculated, and the optimum path coordinates are found in the IP core and sent to the computer. The designed GA IP core is quite flexible and easy to integrate with target applications.

The general structure of the GA IP core designed in this study is shown in Figure 7. The RNGM runs in parallel with the other operators of GA. A 16-bit random number is generated on each rising edge of the FPGA's system clock pulse and this process is independent from the other GA operators. Whenever it is required, a GA operator can use the random numbers generated at the moment. The finite state machines were used to coordinate the transitions between the sequential modules in the GA system.

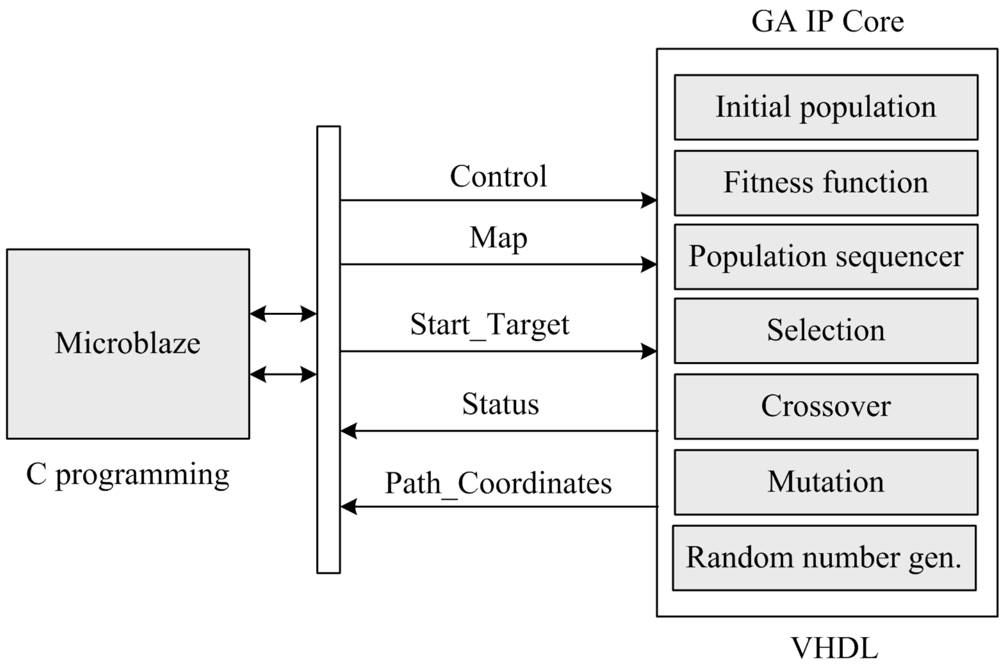


Figure 7. The general structure of the GA IP core.

Table 1 shows the registers and their addresses on the GA IP core. The registers are used to transfer information between the MicroBlaze processor and the IP core. These registers can be thought of as a shared memory. Twenty registers are used, namely Control, Status, Start_Target, Path_Coordinates, and Map registers. The first register's address is Base + 0x00. The purposes of these registers are as follows:

Table 1. The registers and corresponding addresses in the GA IP core.

Address	Register	Write/read	Width (bits)
Base + 0x00	Control	W	1
Base + 0x04	Status	R	1
Base + 0x08	Start_Target	W	16
Base + 0x0C	Path_Coordinates	R	32
Base + 0x10	Map_First_Row	W	32
...	...	...	...
Base + 0x4C	Map_Last_Row	W	32

Map registers: In order to run the GA, the map information first has to be given to the GA. After MicroBlaze takes the map from the camera, it puts the map into the Map registers. In this study, the environment consists of 16 × 16 grids for a map. Therefore, 16 registers are used for each row of the map. With these registers, the map information can be sent to the IP core from MicroBlaze. The first address row of the map is Base + 0x10 and the last address is Base + 0X4C.

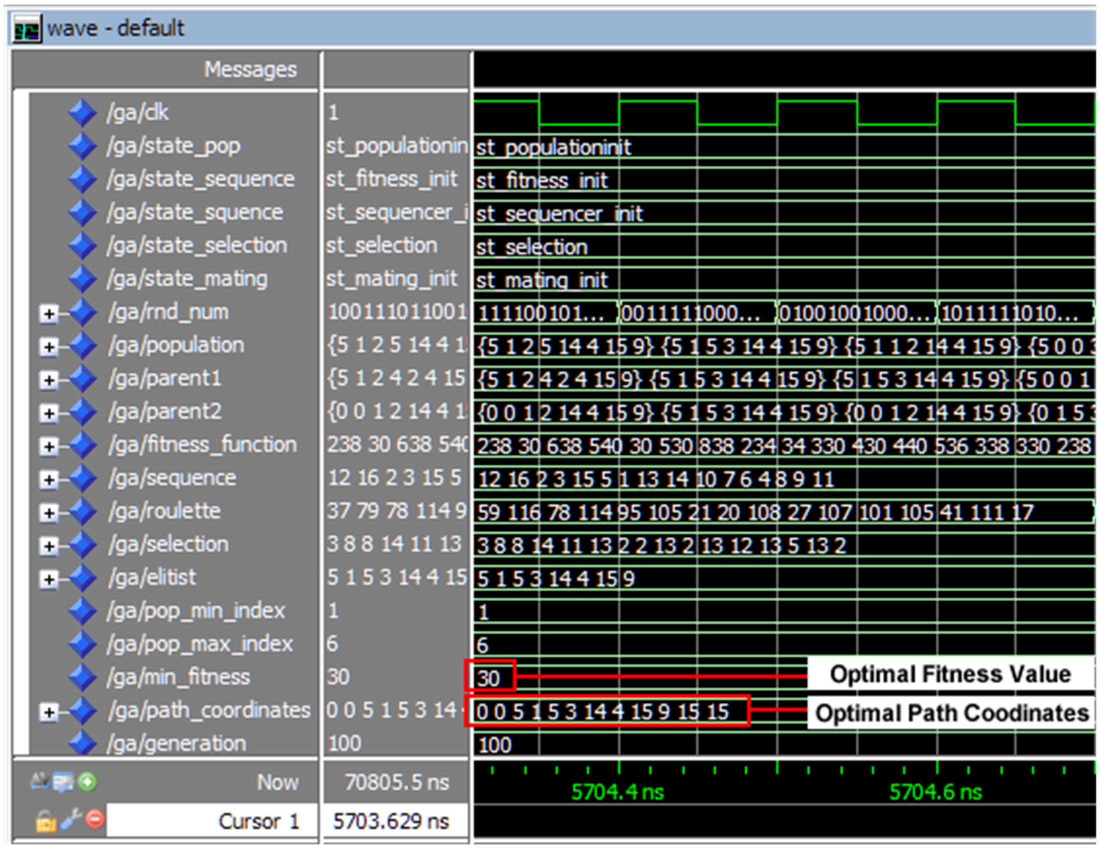
Start_Target register: This register is used to send the coordinates of the start and target nodes of the robot from MicroBlaze to the IP core.

Control register: MicroBlaze informs the GA IP core by means of the Control register when the map is ready in map registers. '0' in the Control register means to *wait* and 1 means to *start*.

Status register: When the IP core finishes the calculation, it notifies the MicroBlaze processor by using this register. While the IP core continues the calculation, the status value is 0. When it finishes the calculation, the status value is 1.

Path_Coordinates register: After the IP core completes its execution, the optimum path coordinates are put into this register and MicroBlaze takes them from here.

Figure 8 shows the values of the GA parameters used in the IP core on the ModelSim screen. On the ModelSim screen, the values of all operators, the best fitness value (the shortest path distance), and the best path coordinates are shown at the end of 100 generations.



```
/ga/clk                 1
/ga/state_pop           st_populationin     st_populationinit
/ga/state_sequence      st_fitness_init     st_fitness_init
/ga/state_squence       st_sequencer_i      st_sequencer_init
/ga/state_selection     st_selection        st_selection
/ga/state_mating        st_mating_init      st_mating_init
/ga/rnd_num             100111011001        111100101...  0011111000...  0100100 1000...  1011111010...
/ga/population          {5 1 2 5 14 4 1     {5 1 2 5 14 4 15 9} {5 1 5 3 14 4 15 9} {5 1 1 2 14 4 15 9} {5 0 0
/ga/parent1             {5 1 2 4 2 4 15     {5 1 2 4 2 4 15 9} {5 1 5 3 14 4 15 9} {5 1 5 3 14 4 15 9} {5 0 0 1
/ga/parent2             {0 0 1 2 14 4 1     {0 0 1 2 14 4 15 9} {5 1 5 3 14 4 15 9} {0 0 1 2 14 4 15 9} {0 1 5
/ga/fitness_function    238 30 638 540      238 30 638 540 30 530 838 234 34 330 430 440 536 338 330 238
/ga/sequence            12 16 2 3 15 5      12 16 2 3 15 5 1 13 14 10 7 6 4 8 9 11
/ga/roulette            37 79 78 114 9      59 116 78 114 95 105 21 20 108 27 107 101 105 41 111 17
/ga/selection           3 8 8 14 11 13      3 8 8 14 11 13 2 2 13 2 13 12 13 5 13 2
/ga/elitist             5 1 5 3 14 4 15     5 1 5 3 14 4 15 9
/ga/pop_min_index       1                   1
/ga/pop_max_index       6                   6
/ga/min_fitness         30                  30
/ga/path_coordinates    0 0 5 1 5 3 14      0 0 5 1 5 3 14 4 15 9 15 15
/ga/generation          100                 100
Now                     70805.5 ns
Cursor 1                5703.629 ns
```

Figure 8. ModelSim screen of the GA parameters.

4. Testing the designed system in a laboratory environment

This study was conducted on the basis of our previous work, in which a software GA was run on a computer to find the optimum path [26]. In this study, a hardware GA IP core on an FPGA was used instead of the software GA on a computer. The Pioneer 3-DX mobile robot and a camera were used in a laboratory environment to confirm the designed GA IP core. Figure 9 shows the laboratory environment, the Pioneer 3-DX mobile robot, and the obstacles used in the study.

The camera is mounted on the ceiling and it sends real-time images of the environment to a computer. Each object in the environment is labeled with a different color: green for the obstacles, blue for the target, and yellow for the mobile robot. In order to determine the heading angle of the mobile robot, it is also labeled with a red color. The direction of the line that connects the centers of yellow and red circles on the robot corresponds to its heading angle.

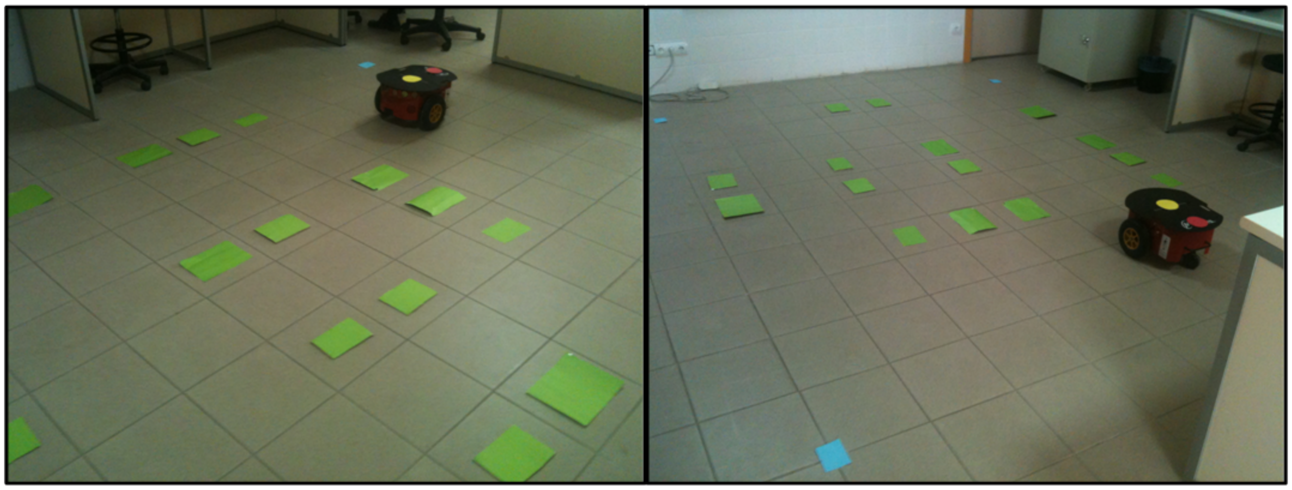


Figure 9. The application environment with a Pioneer 3-DX robot and obstacles.

Figure 10 shows the implementation of the path planning system in a real environment step by step. In the first step, an environment image is captured with a camera. This image is converted into map information by using image processing methods on the computer that is connected to the camera. In the second step, the obtained map is sent from the computer to the FPGA. The GA IP core on the FPGA takes the map information and performs calculations for path planning. In the third step, the coordinates of the determined path are sent to the computer. In the fourth step, the path coordinates are converted into Pioneer 3-DX motion commands on the computer. Then the commands are sent from the computer to the Pioneer robot via RS-232.

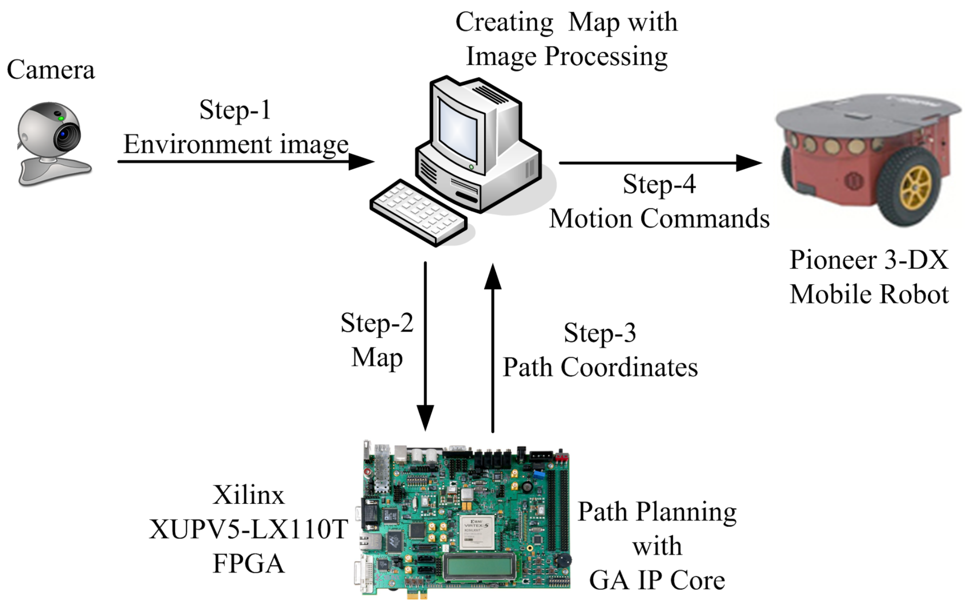


Figure 10. The implementation of the path planning system in a real environment.

5. Experimental results

After the GA IP core was implemented, the processing times of all the hardware GA operators were identified and the performance of the system was evaluated. In order to make a comparison with our earlier study, the processing times of that GA that was written in C codes and works on a computer, the processing times of a GA that was written in C codes and works on a MicroBlaze soft processor on the FPGA, and the processing times of this hardware GA IP core that was written in VHDL and works on an FPGA are given in the Table 2. The population size was taken as 16 chromosomes for all 3 GA systems. The computer used in the experiments has an Intel i5- 2.53 GHz processor and 4 GB RAM. The FPGA used in the experiments has a clock rate of 78 MHz (12.8 ns) for the GA IP core.

Table 2. Process times of the operators of all the 3 GA systems.

GA operators	Software GA (μ s)	MicroBlaze (μ s)	IP Core (μ s)	Improvement (%)
Initial population	2.33	18110	0.82	64.8
Fitness function	30.72	336000	4.10	86.6
Population sequencer	1.33	1860	0.05	96.2
Selection	1.64	5320	0.92	43.9
Mating	0.65	1910	0.05	92.3
Crossover	1.13	3040	0.10	91.1
Mutation	3.47	19640	0.20	94.2
Total time with 100 generations	6.167×10^3	$35{,}000 \times 10^3$	0.594×10^3	90.3

According to the processing times given in the table, the GA that runs on the MicroBlaze soft processor was the slowest system. This result is quite normal because the code that runs on the 125-MHz FPGA is slower than the same code that runs on the 2.53-GHz computer. The GA IP core on the FPGA runs faster than both the computer and MicroBlaze do. The hardware GA provides amazing improvements in process times, which vary between 43.9% and 96.2%, when compared to the software GA that runs on the computer. When the population size is taken as 16 and the generation number as 100, the overall solution time of the shortest path determination process is about 0.6 ms for the IP core and about 6 ms for the computer. As a result, the proposed GA IP core achieves significant improvements in process times when compared to its software version.

6. Conclusion

In this study, the GA was used for feasible path planning for mobile robots in order to find the shortest path. A customized IP core design and the implementation of the GA on an FPGA was proposed. The MicroBlaze soft processor was used in the FPGA as a communication unit between the GA IP core and the outside peripherals. The coding was done with the VHDL and it was simulated to test its correctness on the Xilinx ISE 11.1 platform. The experimental results of verification showed that the GA IP core achieves impressive time speedups when compared to its software version. Then real experiments were conducted using a Pioneer 3-DX mobile robot, a Xilinx xupv5-lx110t FPGA device, and a camera. Within the framework of this study, the entire system was performed in a real laboratory environment so as to demonstrate that the study was applicable in the real world. The hardware GA provided improvements in processing time, which varied between 43.9% and 96.2%, when compared to the software GA that runs on the computer. This result makes the hardware GA the fastest choice for the path planning of autonomous mobile robots, especially for dynamic environments.

Acknowledgment

This work was supported by the Kocaeli University Scientific Research Projects under Project Number 2010/97.

References

[1] Willms AR, Yang SX. An efficient dynamic system for real-time robot-path planning. IEEE T Syst Man Cyb 2006; 36: 755-766.

[2] Tuncer A, Yildirim M. Dynamic path planning of mobile robots with improved genetic algorithm. Comput Electr Eng 2012; 38: 1564-1572.

[3] Manikas TW, Ashenayi K, Wainwright RL. Genetic algorithms for autonomous robot navigation. IEEE Instru Meas Mag 2007; 10: 26-31.

[4] Al-Taharwa I, Sheta A, Al-Weshah M. A mobile robot path planning using genetic algorithm in static environment. J Comput Sci 2008; 4: 341-344.

[5] Deliparaschos KM, Doyamis GC, Tzafestas SG. A parameterised genetic algorithm IP core: FPGA design, implementation and performance evaluation. Int J Electron 2008; 95: 1149-1166.

[6] Fernando PR, Katkoori S, Keymeulen D, Zebulum D, Stoica A. Customizable FPGA IP core implementation of a general-purpose genetic algorithm engine. IEEE T Evolut Comput 2010; 14: 133-149.

[7] Mostafa HE, Khadragi AI, Hanafi YY. Hardware implementation of genetic algorithm on FPGA. In: 21st National Radio Science Conference; 16–18 March 2004; Egypt. pp. 1-9.

[8] Tuncer A, Yildirim M, Erkan K. A hybrid implementation of genetic algorithm for path planning of mobile robots on FPGA. In: The 27th International Symposium on Computer and Information Sciences; 3–4 October 2012; Paris, France. pp. 459-465.

[9] Glette K, Torresen J. A flexible on-chip evolution system implemented on a Xilinx Virtex-II Pro device. In: 6th International Conference on Evolvable Systems; 12–14 September 2015; Sitges, Spain. pp. 66-75.

[10] Gomez-Pulido JA, Vega-Rodriguez MA, Sanchez-Perez JM, Priem-Mendes S, Carreira V. Accelerating floating-point fitness functions in evolutionary algorithms a FPGA-CPU-GPU performance comparison. Genet Program Evol M 2011; 12: 403-427.

[11] Allaire FCJ, Tarbouchi M, Labonté G, Fusina G. FPGA implementation of genetic algorithm for UAV real-time path planning. J Intell Robot Syst 2009; 54: 495-510.

[12] Kok J, Gonzalez LF, Kelson N. FPGA implementation of an evolutionary algorithm for autonomous unmanned aerial vehicle on-board path planning. IEEE T Evolut Comput 2013; 17: 272-281.

[13] Hachour O. The proposed genetic FPGA implementation for path planning of autonomous mobile robot. International Journal of Circuits, Systems and Signal Processing 2008; 2: 151-167.

[14] Holland JH. Adaptation in Natural and Artificial Systems. Ann Arbor, MI, USA: University of Michigan Press, 1975.

[15] Tuncer A, Yildirim M. Chromosome coding methods in genetic algorithm for path planning of mobile robots. In: The 26th International Symposium on Computer and Information Sciences; 26–28 September 2012; London, UK. pp. 377-383.

[16] Gelenbe E, Liu P, Lainé J. Genetic algorithms for route discovery. IEEE T Syst Man Cy B 2006; 36: 1247-1254.

[17] Lysecky RL, Vahid F. A study of the speedups and competitiveness of FPGA soft processor cores using dynamic hardware/software partitioning. In: Proceedings of the Design, Automation and Test in Europe Conference and Exhibition; 7–11 March 2005; Munich, Germany. pp. 18-23.

[18] Xilinx Inc. MicroBlaze Processor Reference Guide EDK 11.4, Xilinx, UG081 (v10.3). San Jose, CA, USA: Xilinx, 2009.

[19] Chen P, Chen R, Chang Y, Shieh L, Malki HA. Hardware implementation for a genetic algorithm. IEEE T Instrum Meas 2008; 57: 699-705.

[20] Serra M, Slater T, Muzio JC, Miller DM. The analysis of one-dimensional linear cellular automata and their aliasing properties. IEEE T Comput Aid D 1990; 9: 767-778.

[21] Scott SD, Samal A, Seth S. HGA: A hardware-based genetic algorithm. In: Proceedings of the Third International ACM Symposium on Field-Programmable Gate Arrays; 1995; California, USA. pp. 53-59.

[22] Wolfram S. Statistical mechanics of cellular automata. Rev Mod Phys 1983; 55: 601-644.

[23] Wolfram S. Universality and complexity in cellular automata. Physica D 1984; 10: 1-35.

[24] Bresenham JE. Algorithm for computer control of a digital plotter. IBM Syst J 1965; 4: 25-30.

[25] Sekanina L. Towards evolvable IP cores for FPGAs. In: Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware; 9–11 July 2003; Chicago, IL, USA. pp. 145-154.

[26] Tuncer A, Yildirim M, Erkan K. A motion planning system for mobile robots. Adv Electr Comput En 2012; 12: 57-62.