AN ENVIROMENT FOR ROBOT SOCCER GAME SIMULATION USING RECONFIGURABLE ARCHITECTURES BASED ON A FPGA-PCI BOARD

Ênio Prates Vasconcelos Filho, enioprates@yahoo.com.br
Gabriel Falcão, gabriel.falcao@gmail.com
Carlos Humberto Llanos, llanos@unb.br

Abstract. *This paper describes the implementation of a platform for simulating the robot soccer game based on the use of reconfigurable architectures. In this case, a FPGA-PCI board was used (based on Stratrix II device, from Altera) in which the strategy of the game for approaching issues (namely, the player approaching the ball) is developed. A game simulator enviorement was also developed using the OpenGL–Graphics Library, which runs in the PC. A communication protocol was defined and implemented in order to support the communication between the FPGA embedded system and the software application. The FPGA embedded system consists of a NIOS embedded processor, in which was implemented a controller system, a memory block, a clock synthesizer (PLL), a timer and a PCI-bus driver. The communication between the simulator and the FPGA-PCI board is achieved by the PCI bus, in which case a communication protocol was defined. The communication among the different FPGA-embedded-hardware blocks was accomplished by means of the Avalon-Bus. The NIOS processor executes several functions for robot soccer game strategy implementation, which were written in C language. These functions are responsible for executing several mathematics operations in order to define the different situations of the game in real time. As far as the authors know, there are no systems in the literature with the similar characteristics (involving reconfigurable architectures, a FPGA-PCI board and a complex embedded system for solving the strategy planning subject for the robot soccer game problem).*

Keywords*: Robot Soccer Game, FPGA, PCI-bus, Embedded Processors.*

1. INTRODUCTION

This paper describes a new architectural approach for the robot soccer game strategy implementation in which specific roles namely attacker, defender and goal-keeper were defined. This approach involves a specific architecture based on a Soccer Game Simulator (SGS), running on a PC (developed on *OpenGL–Graphics Library* and C language), a strategy module (which was developed using a Stratix II FPGA for PCI bus) and a communication protocol for achieving the communication between the simulator and the FPGA-PCI board.

Robot soccer game problem have been popular into the academic environment given that it provides a good platform for multi-agent domain research, dealing with issues such as cooperation by distributed control, effective and fault tolerant communication, real time image processing, real time robotics path planning and obstacle avoidance. Basically, there are three main topics in robot soccer games: (a) the mechanism design of robots, (b) the visual recognition and (c) the study of team strategy planning.

The treatment of computational complex problems has been traditionally carried out with the use of parallel computing. In this model, the computational elements are processors of high performance, however following the von Neumann model. One alternative technique is the use of *Field Programmable Gate Arrays* (FPGAs), which allow describing the algorithms directly in the hardware, using the intrinsic parallelism of these devices. Given that an execution of an algorithm in hardware does not depend of the execution of a set of instructions (as in the von Neumann model) the produced calculations only depends of the critical paths of the implemented circuit. An important point inside of the FPGAs design is the possibility to include embedded complex elements such as processors, Digital Signals Processor (DSPs), hardware for communication, buses, among others. This approach allows the complex solution implementations in an only integrated circuit (System on Chip - SoC).

Another important element inside of the FPGA approach, for high performance solutions implementation, is the use of FPGA based boards for PCI bus. Given the high data exchange rate in this bus, this make possible the implementation of hardware accelerators (for instance, implementing reconfigurable co-processors for specific applications), which allow to improve the performance of the overall computer system. Basically, the PCI protocol defines a structure of master/slave. The master device only can initiate a transaction by means of a solicitation to the arbitrator (the processor that controls PCI bus). The PCI bus is currently seen as the standard of linking multiples peripherals on a PC and it allows transparent upgrade from 32 to 64 bits and working with both 33 and 66 MHz.

In this paper the FPGA is used for the strategy problem solution (namely the speed and the approximation control of the players), the kicking ball strategy and the kinematics aspects of the ball/robots. In this work the players of the adversary team were not taken into account, given that the main aim of this proposal is only the implementation in hardware of the approaching techniques (namely, the player approaching the ball). The strategy module was implemented through an embedded processor, the NIOS from Altera (NIOS II Hardware Tutorial, 2007), an embedded RAM module and using the Avalon bus (Altera, 2006a) for connecting all the architectural elements of the system. The treatment of the problems is adapted for different player roles such as attacker, defender and goal-keeper. These role behaviors are defined and described by several Finite State Machines (FSMs), which define the proper algorithm for each role and game situation. In this approach the Soccer Game Simulator (SGS) is capable to represent the current state of the ball and the robots, apart from the field shape in real time. Additionally, the SGS sends to the Strategy

Module (that was implemented in the FPGA-PCI board) the information about the current game status. The overall strategy module implements a reconfigurable and flexible co-processor for the soccer game, which can be adapted for implementing other strategy techniques.

In order to connect the SGS and the strategy module a communication protocol was defined, which implements a special communication strategy, avoiding communication conflicts. The novelty of this approach is that the strategy tasks of the system were implemented on a FPGA for PCI bus (PCI Development Kit, 2007), achieving a reconfigurable system, which is capable to be easily adapted for new algorithms (using the same architecture). Moreover, this approach also includes a robot soccer game simulator that should be useful to test different game strategies. On the other hand, the use of a dedicated hardware to calculate the robot player strategy opens a wide variety of new possibilities to the robot soccer problem. This approach allows both to change the performance parameters in a flexible way and a faster decision process.

In section 2 the several related works in robot soccer game are discussed. In section 3 the overall architecture of the system is described. Section 4 presents the basic concepts of the proposed embedded architectural system in the FPGA. Section 4 discuses the defined command set for the control system. Section 5 describes the virtual environment for simulating the vehicle motion. Section 6 describes the communication approach. Section 7 describes the project of the robot soccer game simulator and before our conclusions section 8 describes our results.

2. RELATED WORKS

There are several approaches for soccer game strategy problem, which involve the use of artificial intelligent techniques, such as Artificial Neural Networks (ANNs), Fuzzy Logic, among others. An important point is that most of these approaches define several player roles such as attacker, defender and goal-keeper (Hugel *et al.*, 2000). In this case is very important to solve typical problems such as optimizing the trajectory between the robot and the ball, the player speed control, the strategy for kicking the ball, the strategy variation depending of the ball and player positions on the soccer field, among others. In the work of Ming-Yuan *et al*. (2005) a special robot strategy is proposed, depending on the current position of the robot, the ball and the opponents. For instance, at a given time the robot might be closer to the goal or to the ball, in this situation the robot have to adjust its position.

In the work of Bruce, J. *et al*. (2003) are defined both tactics and strategies. The tactics are related to high-level individual robot skills, such as shooting, passing, blocking, etc. The strategies are formulated as plays, where a play is a sequence of tactics assigned to each robot. The robot's speed is regulated by a PID controller system in order to maintain the speed to a defined target.

The use of fuzzy control in robot soccer games has been intensively researched in the field of robot navigation for steering and obstacle avoidance. These techniques have also been applied to the robot soccer for implementing individual robot behaviors and actions, in particular for shooting and obstacle avoidance. In the work of Vadakkepat *et al.* (2004) a fuzzy approach is used to implement individual behavior, to coordinate the various behaviors, to select roles for each robot, speed control, among others. The work of Köse *et al*. (2003) proposes an assign strategy to the robot based on the marked-driven method. To that, different cost functions (score cost functions) are evaluated. In this case the robot with the smallest attacker score cost will be the primary attacker. The most important task is scoring the goal (making the ball enter into the opponent's goal). On the other hand, opponent goals should be avoided. This defines three main roles: *attacker, defender* and the *goal-keeper* (Aranibar, 2005). The main roles can be all represented by a Finite State Machines (FSM). In the work of Chen *et al* (2005) is proposed a robot soccer game approach based on Action Select Mechanism (ASM) implementation. This approach uses Artificial Immune Network (AIN), which is used to carry out the gamming strategy.

An important conclusion about the study of these works is the importance of defining the roles, the strategies and the suitable models for representing the different agents (for example, using FSMs). Our approach defines the same robot's roles (agents) as proposed in the works of Kose *et al* (2003) and Hugel *et al* (2000). Otherwise, our work follows the strategy definitions as suggested in the work of Kose *et al* (2003). Additionally, a motion plan technique was proposed in order to approach the ball position as proposed in the work of Tsung-Ying *et al* (2004). An important point that can be observed in these works is that there are not references in order to optimize the implemented techniques by using a dedicated hardware. In this case the FPGAs can be used as dedicated and flexible coprocessors, suitable for improving the performance of the techniques by implementing strategies directly in hardware (instead of the software implementation) for the robot soccer game problem.

3. SYSTEM GENERAL DESCRIPTIONS

Figure 1 depicts the overall system that was implemented. The SGS graphically represents the game status in real time to the user. Additionally, it gathers the data that will be sent to the strategy module (implemented in the FPGA). These data are sent through the PCI bus by using a defined communication protocol. These data will be treated by the embedded microprocessor NIOS, which was synthesized into the FPGA.

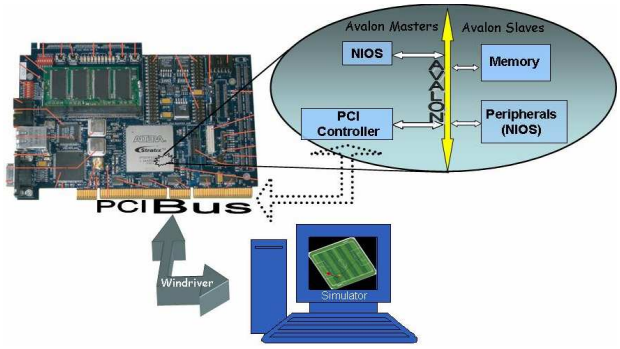


Figure 1. The implemented system

The NIOS produces the appropriated commands for moving the robots in the simulator (SGS), taking into account the implemented strategies. The exchange of information inside of the FPGA chip will be carried through the Avalon bus, which connects the PCI-Controller module, the RAM memory block and the NIOS processor. In order to accomplish the communication between the strategy module and the SGS a set of primitives (written in C language) were created using Windriver software (WinDriver, 2007).

4. THE FPGA EMBEDDED HARDWARE IMPLEMENTATION

To accomplish the calculation in the strategy module and to explore potential of the current development system the following elements were embedded in the FPGA (see Fig. 2):

a) **A NIOS processor**: it is used to control the overall calculation system. The processor receives the state data from the SGS and sends the commands to the robots. The NIOS implements the game strategy, which was written in C language.
b) **The memory block with 128 Kb**: it is an FPGA embedded memory and is used to store both data and instructions that are read by the NIOS. It is also used as *stack* and *heap* for data. Additionally, a memory block is reserved to be used in the PCI communication protocol.
c) **The PLL module (Phase Looked Loop)**: It generates the appropriated clock signals to be used by the embedded processor (50 MHz). The PLL divides the external clock to be used by the peripherals and the NIOS.
d) **The JTAG Driver**: The JTAG is responsible for carrying out the communication between NIOS II and the software environment system allowing the use a console for debugging tasks.
e) **The Timer**: it has the function to provide the time base for the transactions with the JTAG module.
f) **PCI Driver (Altera Intellectual property)**: it allows the implementation (in a friendly way) of the communication between PCI bus and the logic embedded in the FPGA-PCI board.
g) **SysId module**: It implements an identifier number, based on the board hardware configuration, making an identification of the project.

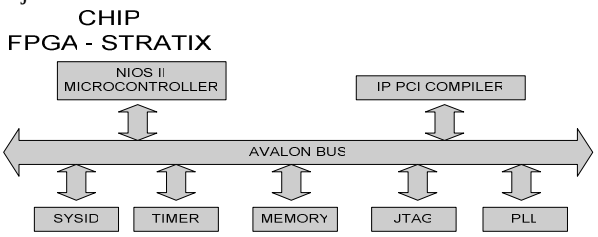


Figure 2. The system architecture to be embedded in the FPGA

All the embedded peripherals are connected to NIOS II and to the PCI controller by means of the Avalon Switch Fabric tool (NIOS II Hardware Tutorial, 2007 and Quartus II, 2007).

5. THE EMBEDDED SOFTWARE

The main objective in a soccer game is the goal and the same point of view is used in the robot soccer. To do that, the player must obtain the knowledge about several factors such as position of the goal, the angle between the ball and the goal center, the current ball status and player's positions with relation to the field, among others.

The software implementation is based on the model with three robots (Köse *et al*, 2004), implementing roles namely *goal-keeper*, *defender* and *attacker*. The goal keeper agent is observed as a player that stays near his own goal line. In this case his only task is to prevent the goal. However, the defender and the attacker are players who change their functions by means of the current best positioning and depending of the necessities. The defender is defined as one who has greater possibility of intercepting the ball. He has to go between the opponent and the ball/goal trying to block them (this role is not described in this paper). On the other hand, the attacker agent task is related to the highest possibility to shoot the ball to the adversary goal and/or to make the goal. Figure 3 shows the control routines for calculating and analyzing the current position and the robots trajectories generation. The software was implemented in C and C++ language. These routines are executed by the NIOS processor and they are explained above.

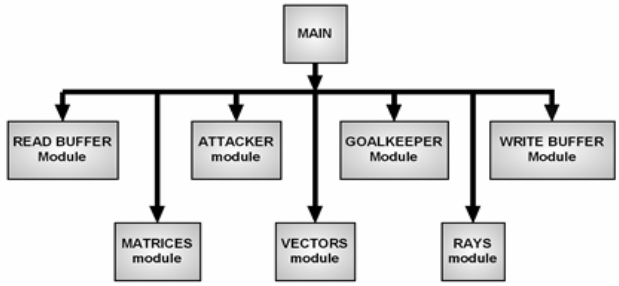


Figure 3. The embedded software in the NIOS

- **Read-buffer module**: this block reads data from a special memory block area. This memory area is used by the SGS to write the status information about the game. The module reads these data and stores them on vectors to be used by the strategy generator.
- **Write-buffer module**: this block writes data to a special memory block area. This memory area is used by the strategy controller to puts the new robot position and speed in the memory, which will be read by the SGS.
- **Attacker and goalkeeper modules**: these modules implement the roles of the players.
- **Storing-vector module**: this module implements the mathematical functions that are used to store the robot data such as robot's orientation and speed.
- **Rays module**: This module implements the mathematical functions that allow the calculation of distances between points and vectors.
- **Matrix module**: This module implements some matrix operations.

The different modules were developed inside the NIOS II IDE tool. There were included some libraries in the project in order to calculate matrices, vectors, rays and angles from positions and specific points. These libraries were also used in the SGS design.

During the project development, two strategies were created to determine the trajectory of the *attacker* agent. The first one calculates his positioning taking into account some divisions in the field and the angle between the robot and the center of the ball. This strategy was called as *Reference Lines Algorithm* (RLA) and it divides the field using some lines, which are used as references to the robot positioning. The second algorithm determines a straight line between the goal center and the center of the ball. From this line, the processor calculates where the attacker should intercept the ball in order to take it into the adversary goal. This second algorithm showed itself more efficient than the first one (see section 8). This method was called as *Line Ball/Goal Algorithm* (LBGA).

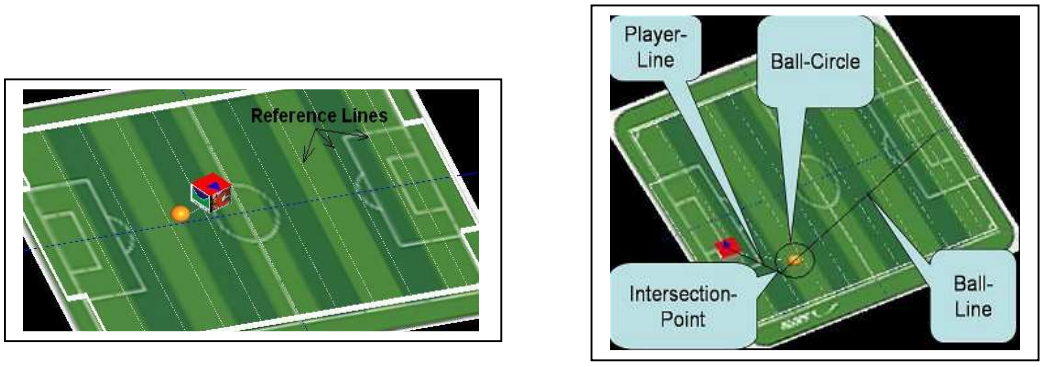
Only one strategy was developed for the *goal keeper* agent. It is important to observe that the behavior of the goal keeper is mostly preventive, calculating the future ball's position for intercepting it before the ball crosses the goal line. On the other hand, the attacker has to approach the ball only from its current position.

5.1. Trajectory Calculation from the Attacker using the reference lines (the RLA algorithm)

The game field was divided in zones under the *X* axis (see Fig. 4.a). This divisions were defines in the *line-vector*: {-200, -160, -80, -40, 0, 40, 80, 120, 160, 200}. The RLA algorithm determines the position from the ball with regard to the next line and creates a trajectory that allows the attacker to put the ball into the goal. The steps of the algorithm are the following:

1) The controller determines if the *X* position from the center of the attacker agent has a coordinate smaller than the ball (if the attacker is behind the ball).
2) If the attacker is in front of the ball he goes back for achieving a better position.
3) It is calculated a line between the center of the goal and the center of the ball.

4) The controller begins to approach the ball, calculating in real time the current distance between the ball and the attacker agent. If this distance is bigger then a threshold, the attacker agent continues his approximation to the ball (adjusting his angle).
5) When the distance between the attacker and the ball is smaller than the threshold his objective becomes to approach the ball with an angle that permits to kick the ball to the goal.
6) When the angle is found the attacker agent kicks the ball to goal.



(a) (b)

Figure 4. (a) The simulated field and the reference lines (b) The elements of the attacker agent algorithm

5.2. Trajectory Calculation from the Attacker – Line Ball/Goal (the LBGA algorithm)

The second developed algorithm (LBGA) works with the relationship between the ball and the goal.

1) The controller determines if the *X* position from the center of the attacker agent has a coordinate smaller than the ball (if the attacker is behind the ball).
2) If the attacker is in front of the ball he goes back for achieving a better position.
3) It is calculated a line between the center of the goal and the center of the ball (*the ball-line*, see Fig. 4.b).
4) It is defined a circle (the *ball-circle*, see Fig. 4.b) around the ball whose radio is defined by a threshold.
5) It is calculated a line (the *player-line*, see Fig 4.b) between the player and the intersection of the *ball-line* and the *ball-circle* (the *intersection-point*).
6) It is calculated the angle in which the robot must approach the *intersection-point* as well as the distance between robot and the *intersection-point* (in real time). This data defines the attacker-trajectory.
7) The robot should be in the *intersection point* whenever the distance between the attacker and the *ball-circle* is smaller than the threshold. If the interception-point was not accomplished it is defined the best possible angle to the robot.
8) The ball is kicked.

Figure 5 shows a Finite State Machine (FSM) of the attacker agent. The transitions mean events in the attacker agent process and are the following:

- *E1*: the *X* position of the robot is bigger than the *X* position of the ball: *X_RobotPosition* > *X_BallPosition*.
- *E2*: the *X* position of the robot is smaller than the *X* position of the ball: *X_RobotPosition* < *X_BallPosition*.
- *E3*: the kick angle of the ball is smaller that the pre-defined limit.
- *E4*: the distance between the robot and the ball is smaller than the threshold.

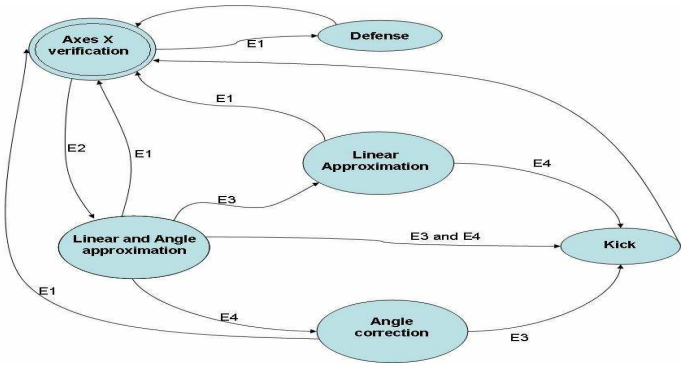


Figure 5. The FSM of the attacker – LBGA algorithm

Figure 6 shows the zones in which the attacker-agent is more efficient (in blue color). The more difficult positions are shown in red-color. That happens because the ball's position produces a point out of the field and they are difficult to be intercepted, in accordance with the algorithm (for instance near the side lines or the corners).

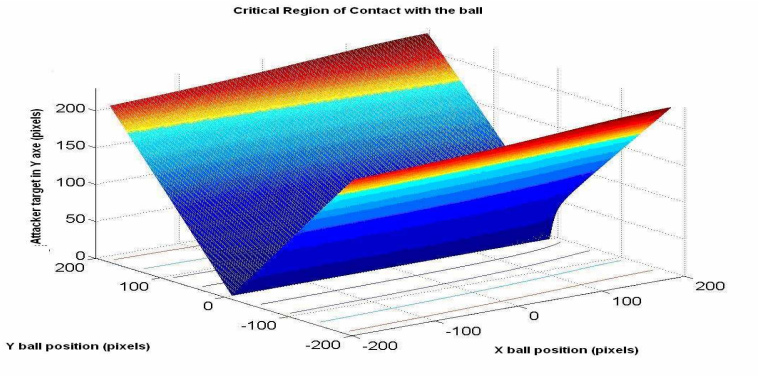


Figure 6. The critical region from goal score

6. THE COMMUNICATION PROTOCOL

A protocol was developed in order to implement the communication between the PC and the FPGA Board. The protocol uses the RAM memory block for storing the data coming from the SGS. The NIOS read this status information and writes the proper commands in a specific memory address. Thus, the protocol will send the commands to the simulator environment at an appropriate time. The reading and writing process were carried out without any conflicts. Special memory addresses were created inside of the memory to implement the suitable writing/reading permissions. A given memory word represents the TX_ACK_NIOS tag and another one is used for representing the RX_ACK_NIOS one. Similarly, the TX_ACK_SIMULADOR and RX_ACK_SIMULADOR tags are memory mapped as well. Two programs were written for implementing the communication protocol between the FPGA and the SGS. Figure 7 shows the FSM representing the implemented protocol in the NIOS processor. A similar program was implemented in the simulator (SGS program).

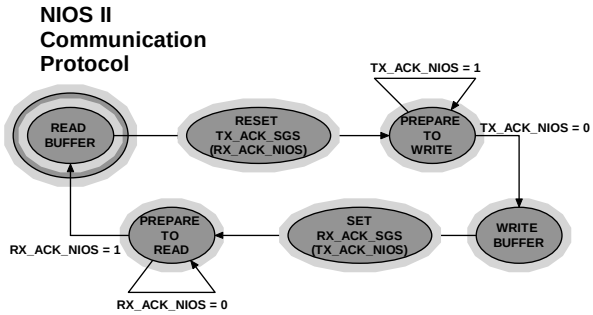


Figure 7. The FSM of the NIOS protocol

The NIOS II protocol begins the process by reading the read-buffer (see Fig. 7). Therefore, the robot and the ball positions are known (the *read-buffer* state). Then the algorithm resets the TX_ACK_SGS tag; this is similar to the RX_ACK_NIOS tag (the *reset* state). This means that the embedded software has finished the lecture process and that the SGS can now write again in this memory position. Then the protocol goes to the *prepare-to-write* state. In this state, the machine verifies if the TX_ACK_NIOS tag is equal to '0'. This represents the fact that the SGS has already read the last data and that the embedded software is now able to write in memory. If the TX_ACK_NIOS is equals to '1', the FSM continues waiting until the last data is consumed by the SGS. Then the FSM writes in memory the new robot's positions and speed. Afterwards, the FSM set the RX_ACK_SGS tag to '1' (the TX_ACK_NIOS tag is set as well), indicating that the new data is ready to be read by the SGS (the *set-RX-ACK-SGS* state). Finally, the NIOS II software goes to the *prepare-to-read* state. This state verifies if the SGS has already written the current position of the robots and the ball (this state is similar to *prepare-to-write* state). If the RX_ACK_NIOS tag is equals to '0' the FSM waits until the information is ready. If the RX_ACK_NIOS tag is equals to '1' the protocols restart the process of the data reading.

7. THE SGS PROJECT

In order to implement the SGS the DEVC++ free software was chosen (Bloodsheed Software, 2007). Given that the simulator demands multiple operations with images, the OpenGL-Graphics Library (OpenGl Documents, 2007) was

included (this library was developed in C/C++). The simulator must represent the physical effects of the collisions between *ball/robot*, *robot/wall* and *wall/ball*. Moreover, the SGS must answer to the commands coming through the PCI bus (in a fast way). A treatment of the collisions was developed as well as the conditions of the restrictions for the ball's movements and of the robot in real time. The structure of the software of the simulator is shown in Fig. 8 and the module descriptions are the following:

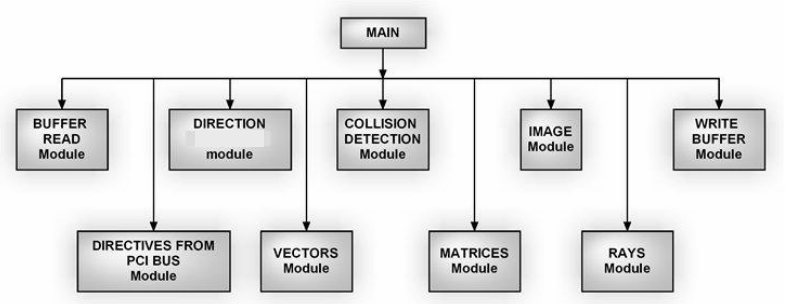


Figure 8. The simulator structure

- *Vectors*, *rays* and *matrices* modules have the same functions as was described in section 5.
- *Buffer Read* module: it communicates with the PCI bus, reading from the memory the future position of the robot and its speed.
- *Direction* Module: the SGS determinates the current direction of the robot.
- *Collision Detection* Module: the SGS determines whether in current movement some collision will occur.
- *Image* Module: the Open-GL is used to print in the monitor the field, the robot and the ball pictures.
- *Write Buffer* Module: the SGS calculates the new positions of the components, based on its interactions during the last movement and writes it in the memory, using the PCI bus.
- *Directives from PCI Bus* Module: this block is based on the PCI driver, whish was generated by the WinDriver software (WinDriver, 2007). This driver provides the basic functions to write and read through the PCI bus and how these functions can be used.

7.2 – Collisions Treatment in the SGS

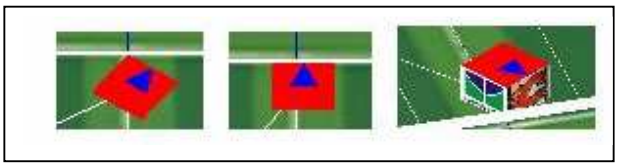
The generation of the movement of the robots inside of the simulator was developed through several vector operations, which represent the speed and the angle variations (the current and previous ball/robots positions are also represented by vectors). The robot movement representations (such as acceleration and the increase of the displacement) were achieved in a realistic way. Additionally, the calculation of collision between the robots and the ball with the walls were obtained.

7.2.1 – The algorithm for detection the robot × walls collisions

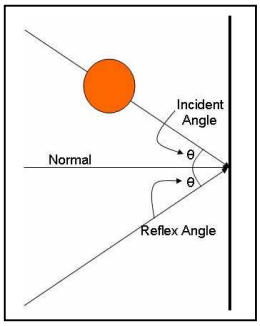
This algorithm has the following steps:

- The angle and the robot speed are obtained.
- The new position of the four vertices of the robot is calculated.
- If none of the vertices collides with any of the walls, the movement is allowed and the algorithm restarts.
- Otherwise, the movement is not allowed and the robot is motionless for that action.

Notice that these calculations only are true because the field comprises only four walls with angles of 90º between them. Therefore, the robot always collides with some of its vertices. Examples of collision between the robot and the walls are showed in Fig. 9.a and Fig. 9.b.



(a)



(b)

Figure 9. (a) The robot × wall collision (b) The ball reaction after collision

7.2.2. The algorithm for detection the ball × walls collisions

This algorithm has the following steps:

- The angle and the ball speed are calculated.
- It is calculated if the direction of the ball is parallel to the wall. In this case there will not have collision and the algorithm restarts.
- In the case of the ball's direction produces an interception with the wall, the algorithm calculates how many time will spend until the collision happens.
- After the collision, the reaction in the ball is defined using the motion equations. In this case the reflection angle is the same of the angle of incidence in relation to the normal line in the surface (see Fig. 9.b).

This calculation is carried out for the walls in an individual way. Therefore, 12 calculations of intersection between balls and walls are done. In the case that the ball has more than one possibility of collision, the time is the definitive factor. That is, the SGS is capable to determinate with what wall the ball will first collide.

7.2.3. Algorithm for detection the Ball × Robot collisions

The SGS assumes that the ball have a quart of the weight of the robot. In this case the movement equation from the ball speed after a collision with a robot is given by eq. 1 (notice that BTC means *before the collision*).

$$BallSpeed = BallSpeed\_BTC \times 0.25 + RobotSpeed \times 1 - (BallSpeed - RobotSpeed)/(1.25) \quad (1)$$

The algorithm is the following:

- The angle and the speed of the ball and the robot are defined.
- The angle between ball and robot and the collision point are calculated. A normal line to this side is calculated for defining the reaction of the ball.
- The relative speed vector between the ball and the robot are defined.
- It is verified if the final position of the ball will have an intersection with the robot. The collision will not happen if there is not an intersection. In this case the algorithm restarts.
- If an intersection exists, the time in which the collision will happen is calculated.
- It is verified if the next collision is with a robot or with some of the walls.
- In the case of a collision with a robot, the equations of movement conservation are used, having as reference the angles of the collision and in which point of the robot the collision will occur.

8. RESULTS

In this section some results of this approach are described.

8.1. The hardware results

Table 1 shows the results for the main modules of the synthesized system using the FPGA Altera device EP1S25F1020C5 and the Quartus II tool.

Table 1. The use of the main FPGA resources elements

System Elements	***Logic Cells***	***Memory Bits***	***DSP Elements***	***DSP 36x36 bits***
System (except *top-level*)	3960	1129344	8	1
Clock 0	104	0	0	0
Clock 1	203	0	0	0
NIOS II	1858	66944	8	1
JTAG Module	162	1024	0	0
PCI Compiler	1202	12800	0	0
PLL	24	0	0	0
On-Chip Memory	2	1048576	0	0
SysId	2	0	0	0
Timer	130	0	0	0

Important to point out the presence of microcontroller NIOS II embedded in the FPGA, resulting in a high consumption of logical components. Moreover, the NIOS II implementation includes a block of memory of 128Kb, in

which only 54Kb are effectively used by the embedded software. Figure 10 shows the block generated by the SOPC Builder tool (from Altera), used in Quartus II with all the connections.

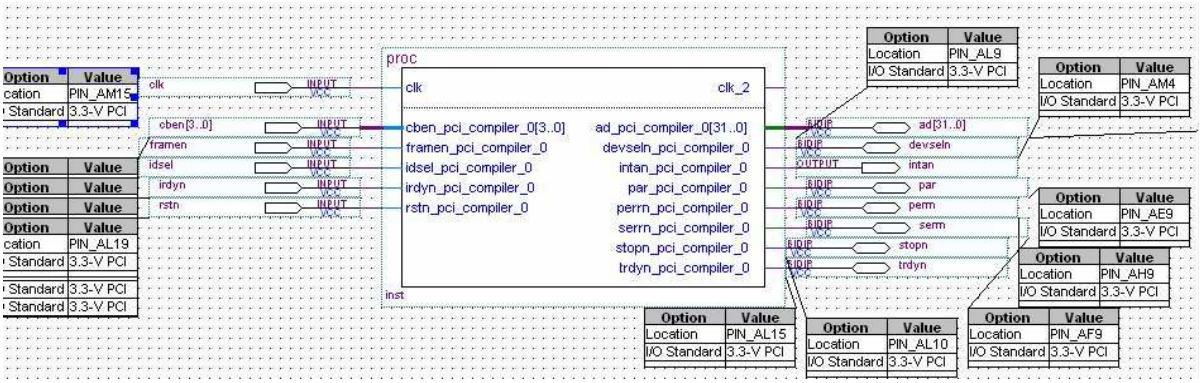


Figure 10. The implemented block inside the FPGA

8.2. The Trajectory Calculation

The calculation of the efficient trajectory for a robot on the table was developed using the NIOS processor. This has the capacity to become a *goalkeeper agent* or an *attacker agent*. This occurred in function of the current development of the simulator. Anyway, the current result shows the viability for using the capacities of calculation of the FPGA for the trajectory planning.

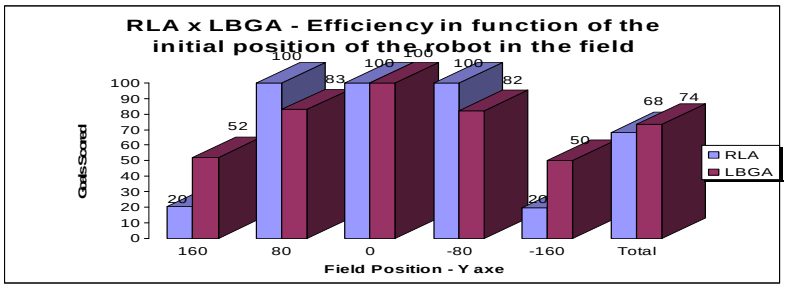


Figure 11: comparing the RLA and LBGA algorithms

8.3. The Speed Data Treatment

The comparison between the two attacker algorithms can be observed in Fig. 11. This figure shows that (for the same initial position in *Y* axis) the LBGA is more efficient than the RLA, in function of the scored goals. This can be observed in the last column, where the LBGA shows a general efficiency of 74% against 68% of RLA algorithm. Anyway, in some specific situations in the field, the RLA is better than LBGA. This is in the case of the positions -160 and +160 (both in *Y* axis).

With regards to the timing for writing/reading data to/from the embedded memory, it was observed that for each reading treatment and writing processes of the NIOS 6 iterations of the simulator are executed (on average). Given that the clock of PC processor is about 3,00GHz, it can be observed as a good result. Even taking into account that the best cameras of the market send about 30 frames for second, the result is still more interesting. This is because of the speed of the simulator was measured in 40 frames per second on average. This is a good indication that our control system will work correctly in a real robot soccer game.

9. CONCLUSIONS

This paper described the implementation of an environment for the robot soccer game problem. In this approach the strategy control was implemented in a reconfigurable architecture based on a FPGA-PCI board. In this case, several devices such as microprocessor, memory board, timers, among other were embedded in the FPGA. This fact defines a System on Chip (SoC) implementation of our controller design approach. Additionally, a soccer game simulator was implemented, which was written using free software tools. A protocol was defined for allowing the communication between the simulator (SGS) and the FPGA embedded system.

Several synthesis results were discussed in section 8. The embedded microprocessor spent about 20% of the FPGA resources. The performance of the strategy control can be improved by implementing the critical function directly in hardware.

A comparative study was accomplished about comparing the two algorithms for the attacker agent. This shows the capability of the system to implement several strategies. The implemented architectures are very promissory given the transference rate of the PCI bus, which is very suitable for implementing reconfigurable co-processors.

Another important result of this approach was that the communication between the different hardware components with the PCI bus was satisfactorily accomplished. As far as the authors know, there are no systems in the literature with the similar characteristics (involving reconfigurable architectures, FPGA-PCI board and the complex embedded system for solving the strategy planning subject for the robot soccer game problem).

The use of FPGA as accelerator devices in supercomputer is currently more and more studied. In this case is very important to connect the FPGA board using high performance buses such as PCI and PCI-Express ones. In Fergusom (2007) is discussed the implementation of the most efficient supercomputer (nowadays), which uses several FPGAs as accelerator hardware devices. This fact shows the importance of the FPGA devices for high performance computation issues.

10. REFERENCES

Aranibar, B. D., 2005, “Estratégias baseadas em aprendizado para coordenação de uma frota de robôs em tarefas cooperativas”. Master work document. Departamento de Eng. de Automação e Computação. Universidade Federal do Rio Grande do Norte, Brasil.

Bloodsheed Software, 2007. Avaliable at http://www.bloodshed.net/dev/. Accessed in 2007.

Bruce, J., Bowling, M., Browning, B., and Veloso, M, 2005, “Multi-robot Team Response to a Multi-robot Opponent Team”. In IEEE International Conference on Robotics anda Automation. Proceedings. ICRA ’03, volume 2, Taipei, Taiwan, pp. 2281-2286.

Chen., C. Li., C., Hsu, W. Wang, Y., (2005), “Design an Action Select Mechanism of Soccer Robots System Using Artificial Inmune Network”. Proceeding of the IEEE International Conference on Mechatronics, pp. 445 -450.

Fergusom, T., 2007. Avaliable at <http://news.zdnet.co.uk/hardware/0,1000000091,39286409,00.htm>. Accessed in 2007.

Hugel, V. Bonnin, P., Blazevic, P., 2000, “Reactive and Adaptative Control Architecture Designed of Sony Legged Robots League in RoboCup 1999”. Proceedings of the 2000 IEEE/RSJ. International Conference on Inteligent Robots and System, pp. 1032- 1037.

Köse, H., Tatlidede, U., Mericli, C., Kaplan, K., Akin, H. L., 2004, “Q-Learning based Market-Driven Multi-Agent Collaboration in Robot Soccer”. Turkish Symposium On Artificial Intelligence and Neural Networks, pp. 219-228.

Ming-Yuan. S., Juing-Shian C., Tien-Lung, Y., Ke-Hao, C., Sheng-Pao, C., 2005, “System Desing and Strategy Integration for five-on-five Robot Soccer Competition”. Proceeding of the 2005 IEEE International Conference on Mechatronics. July, pp. 461 - 466.

Nios II Hardware Tutorial, 2007, San Jose, Ca: Altera. Available at <http:// www.altera.com/>. Accessed in 2007.

Quartus II, 2007, “Version 6.0 Handbook”, San Jose, Ca: Altera. Available at <http:// www.altera.com/>. Accessed in 2007.

PCI Development Kit, 2007. Stratix Edition, Getting Started User Guide, San Jose, Ca: Available at <http:// www.altera.com/>. Accessed in 2007.

Tsung-Ying, T., Chiu-Hao, L., Yyin-Tien, W., 2004, “Term Formation Control for Soccer Robot Systems”. Proceedings of the 2004 IEEE International Conference on Networking, Sensing & Control, pp. 1121- 1125.

Vadakkepat, P., Miin, O., Peng, X., Lee T. H., 2004, “Fuzzy Behavior-Based Control of Mobile Robots”. In IEEE Transactions on Fuzzy Systems, Vol. 12, No. 4, pp. 559- 564.

OpenGl Documents, 2007. Available at <http:// www.opengl.org./>. Accessed in 2007.

WinDriver, 2007. PCI/ISA/CARDBUS v8.02 User’s Guide. Avaliable at <http://www.jungo.com>. Accessed in 2007.

11. RESPONSIBILITY NOTICE

The authors are the only responsible for the printed material included in this paper.