ICROS-SICE International Joint Conference 2009
August 18-21, 2009, Fukuoka International Congress Center, Japan

Modeling and Implementation of Automatic System for Garage Control

Valery Sklyarov, Iouliia Skliarova, Abílio Neves

Department of Electronics, Telecommunications and Informatics,
University of Aveiro/IEETA, Portugal (Tel : +351-234-401-539;
E-mail: skl@ua.pt, iouliia@ua.pt, a31168@ua.pt)

Abstract: The paper describes a system for garage control providing for automatic parking of arriving cars and driving them to the garage exit on requests. The system is composed of two sub-systems that are directly linked for simulation purposes and communicate through a wireless interface needed for physical implementation. One of the basic modules providing for management and priority-driven selection of parking slots is considered in detail. The complete system prototype was designed, implemented in FPGA, validated, and tested. Functional simulation of the designed system is presented in a virtual mode enabling the relevant results to be evaluated visually on a monitor screen without the need for an expensive physical environment.

Keywords: Embedded system, automatic parking, FPGA, remote control, visual simulation

1. INTRODUCTION

The paper is dedicated to the design, implementation in field-programmable gate arrays (FPGA), and modeling of a distributed embedded system for garage control. It is known that the majority of developed digital systems are embedded [1] and they are organized on the basis of one or more computational units that perform all necessary processing work using built-in processors (processing elements – PE). A typical PE receives instructions and data from memory, executes the instructions in accordance with a predefined control sequence (a micro-program) generated by control circuits, and writes the results to memory [2]. The latter can be considered as an external RAM or internal registers.

The approach considered here, instead of mapping a given problem onto a set of PEs with classical organization [2], relies on incremental decomposition of the proposed system architecture and a top-down technique allowing to implement all primarily architectural components in hardware from specification in a hardware description language (HDL) according to predefined models and HDL skeletal fragments called templates.

As mentioned above the system is distributed in a sense that it is organized as a composition of sub-systems communicating through established interfaces. One sub-system is responsible for the garage control, i.e. for the set of such operations as: processing and indication of the most preferable slot for parking any new car on the entrance; opening/closing the gates, providing instructions for parked cars that have to be retrieved, etc. Other sub-systems are installed inside cars and they instruct cars how to drive to the slots indicated by the first sub-system.

Informally the basic functionality of the entire system can be described as follows: any incoming car is stopped in front of the entrance gate where there is an eventual queue in case if there are no free parking slots in the garage. A driver leaves the car and then the automatic system is responsible for further steps of parking. As soon as there appears a free parking slot, the entrance door is opened and the central sub-system instructs the car how to drive to the designated parking slot. In other words, if there exists more than one parking slot, the system selects such one that is more preferable for parking, i.e. which has the highest priority. Requests for retrieving cars from the garage are formed by car owners. The selected car drives automatically to the exit gate (according to the instructions given by the central sub-system) and then is parked in a place where the car owner waits. The entrance/exit gates are opened and closed automatically, which is controlled by the central sub-system.

Thus, the central sub-system is responsible for the following basic operations:

- Opening/closing the garage entrance/exit gates;
- Receiving and processing information about free parking slots;
- Instructing sub-systems installed in cars through an established interface.

A car sub-system is responsible for the following basic operations:

- Automatic driving to the garage (to the indicated parking slot) and from the garage (from the requested parking slot);
- Avoiding any obstacle (*e.g.* walls and other cars) and, thus, implementing intelligent parking control;
- Avoiding any deadlock, for example, arriving cars cannot block cars leaving the parking slots.

This paper focuses on the particular proposed architecture of the entire system, its subsystems and basic architectural components. It does not describe any method for evaluation and comparison of alternative design ideas and models. Although the considered technique is based on HDL skeletal fragments, the most important of which are design templates for advanced finite state machines (FSM), the models and design methodologies for such FSM are also not a target of the paper.

The remainder of this paper is organized in five sections. Section 2 describes problem specification and basic system functionality. Section 3 outlines the design and implementation of the central sub-system. Section 4 is dedicated to sub-systems for controlling individual cars. Section 5 summarizes the implementation details and presents the results of experiments. The conclusion is given in Section 6.

2. PROBLEM SPECIFICATION AND BASIC SYSTEM FUNCTIONALITY

Fig. 1 presents interactions between the two sub-systems introduced in section 1.

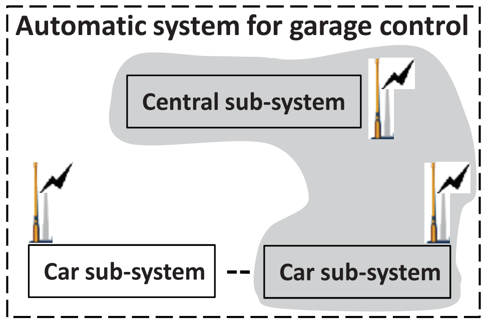
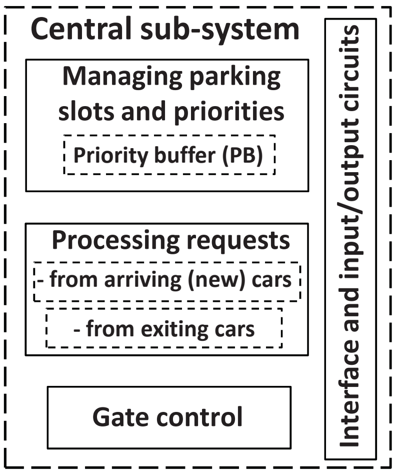


Fig. 1 General structure of automatic system for garage control.

Gray highlighted area indicates the sub-systems that have been designed, implemented and tested. In particular, for the case of wireless communications only interactions between the central sub-system and the sub-system for just one car have been examined and validated. Virtual simulation has been done for a system with many cars (see section 5).

Fig. 2 depicts the proposed architecture of the central sub-system.



Central sub-system
Managing parking slots and priorities
Priority buffer (PB)
Processing requests
- from arriving (new) cars
- from exiting cars
Gate control
Interface and input/output circuits

Fig. 2 General architecture of the central sub-system.

There are four architectural components that are responsible for the following operations:

- Managing parking slots and priorities based on accumulation of data about free parking slots with their priorities for potential parking of (new) arriving cars. The accumulation is done in a priority buffer (PB), which takes input data items about released slots and outputs items with the highest priority. The PB is organized in such a way that the established priorities might be dynamically rearranged as well as some items can be removed on external requests (for example, when some parking slots are reserved for special purposes);
- Processing requests enable the system to output a sequence of instructions for arriving and exiting cars. The sequence of instructions is presented like the following: wait for opening the entrance gate; drive straight to a position A (until getting a signal from the sensor A), turn right; drive straight to a position B, turn left; drive until the slot with the number *i*, turn left and park the car to the slot *i*;
- Control of gates enables entrance and exit gates (doors) to be opened and closed;
- Interface and input/output circuits provide for wireless communication with car sub-systems and interaction with sensors and actuators.

Let us discuss now how to manage priorities of the slots. In the simplest case any slot has a fixed priority based on the proximity to the entrance gate. One possibility for a garage with a fixed number of such slots would be an *array* of bits with each element corresponding to a slot and recording the slot state (*e.g.* 0 – the slot is empty and 1 – the slot is occupied). To allocate a slot would just require searching the array sequentially from the beginning for the first vacant position (see Fig. 3, a). Creating a *list* of vacant slots in priority sequence can be seen as another possible alternative (see Fig. 3, b, c).

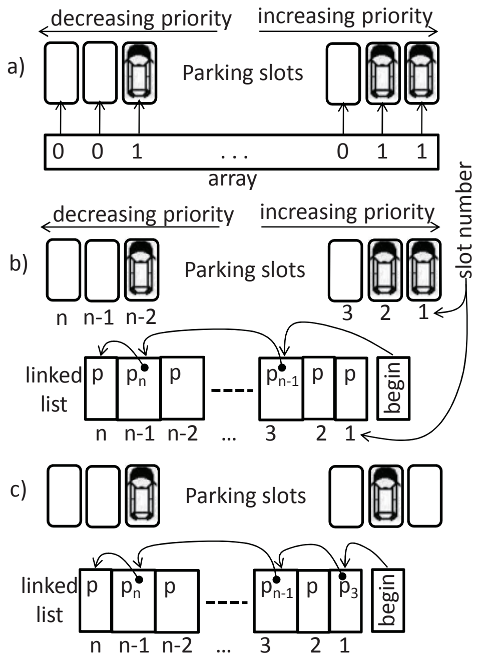


Fig. 3 Using arrays (a) and linked lists (b,c) for a priority buffer.

The number of elements in the list for a garage with a fixed number of slots can be equal to the number of slots (see Fig. 3, b). The initial address of the list is stored in the «*begin*» register. An address field allocated for each element indicates the next most priority-driven vacant slot. Since the number of elements is fixed and they are associated with the relevant slots, each element might contain just a field addressing the next slot. Fig. 3, c shows simple changes that have to be done in the list as soon as the car from the slot 1 is retrieved.

The PB considered here allows more complex management to be provided in such a way that:

- Dynamic changes in priorities of the slots are supported;
- The number of slots is not fixed.

Dynamic changes are useful for many practical situations. Indeed, the considered above priority allocation rule is very simple and in practice it might be significantly more complicated. For example, the necessary location could be determined based on the expected duration of using the garage. The car for someone parking for an hour or two might be allocated a slot close to the entrance (or possibly close to the exit to minimize retrieval time). Someone parking for a week might be allocated a slot as far away as possible from the entrance/exit. It is also possible that a business might book space and pay extra for faster retrieval. There are obviously also many questions about deadlock avoidance that depend on the physical layout and possible routings for cars within the garage. Thus, in general case the priorities cannot be assigned statically. We assume that a dynamic priority management has been implemented in the central sub-system and the PB has to be able to indicate the most priority-driven slots taking into account any dynamic change. A non fixed number of slots might require dynamic memory allocation and de-allocation. If this is not desirable in hardware we can fix the maximum possible number *n* of slots and provide for management of only actually available slots (their number cannot exceed the fixed maximum *n*).

We found that the best data structures allowing dynamic priority changes to be rapidly implemented are *linked lists* (see Fig. 3) and *binary trees* [3]. This is because all necessary changes can easily be done through modifications of pointers that address items associated with parking slots. Arrays shown in Fig. 3, a, can also be used. Suppose there are *n* cells of memory available with a priority indicated for each cell. When the number of slots is less than the maximum *n*, all slots not in the garage can be set as "unavailable". Sequential search for a free slot with the highest priority would be a loop with *n* iterations at maximum, which is appropriate for the considered problem. Implementation of linked lists and binary trees requires more sophisticated circuits but the search over these structures is faster.

Fig. 4 gives an example of linked lists. Let us consider a garage in Fig. 4, a, in which all parking slots are numbered (1,2,3,…,28). The priority of a slot with the number *i* is written inside the relevant rectangle *i* and B indicates «blocked» or «unavailable». Gray rectangles mark occupied slots. The *begin* pointer (see also Fig. 3, b,c) addresses the slot with the highest priority (the smaller the number in rectangle *i*, the higher is the priority of the slot *i*). Suppose some dynamic priority changes have taken place (see bold numbers inside the rectangles in Fig. 4, b). In this case some addresses in the linked list have to be altered and this is indicated by bold numbers on the right-hand part of Fig. 4, b.

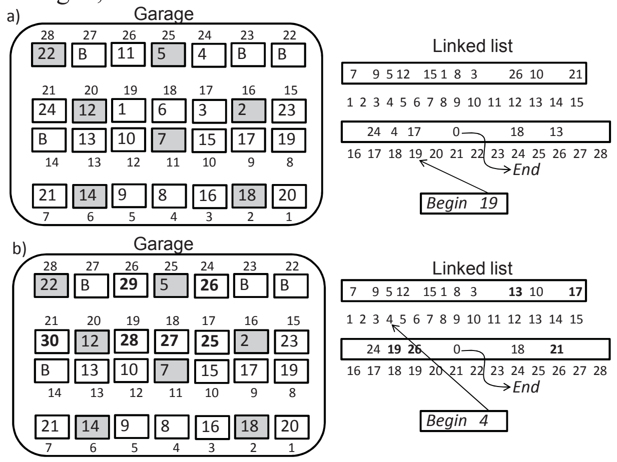


Fig. 4 Using linked lists to manage priorities.

Fig. 5 shows how binary trees can be used for the same example. Let us examine Fig. 5, a. Any node of the tree has a slot number written near the relevant circle and a priority written inside the circle. The tree is built in accordance with [4] in such a way that at any node, the left sub-tree contains only values that have higher priority than the value at the node, and the right sub-tree contains values that have lower priority. The number of iterations enabling the most priority-driven value to be found cannot exceed the longest way from the root to leaves and it is equal to the depth of the leftmost sub-tree (for example, 4 iterations are required in Fig. 5,a: 20-16-8-3-1). This is significantly faster than for arrays but slower than for linked lists.

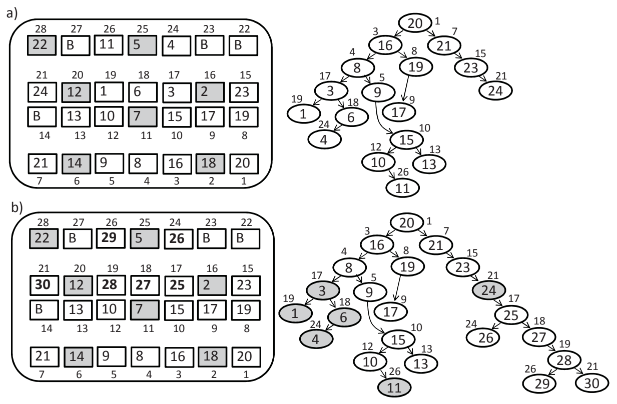


Fig. 5 Using binary trees to manage priorities.

Assuming some priority changes shown in Fig. 5, b, the tree has to be rebuilt as shown on the right-hand part of Fig. 5, b. Gray nodes correspond to the slots whose priorities have been altered. They have to be replaced (see new nodes in Fig. 5, b, with the same numbers as the numbers of the deleted gray nodes). The final new tree is depicted in Fig. 6. All necessary operations can easily be performed applying the technique [3]. Indeed, the nodes representing slots with altered priorities have to be removed from the tree, and the same nodes have to be inserted again to the tree with new priorities. These operations can easily be executed applying modules from [3] that remove nodes with indicated numbers and add designated nodes.

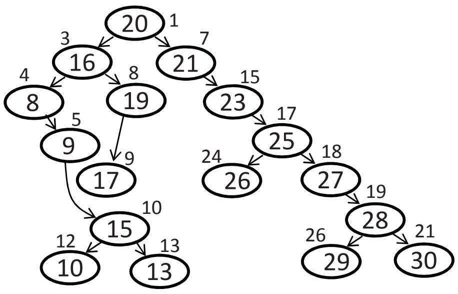


Fig. 6 New tree after dynamic changes.

The PB for the central sub-system can be constructed using any data structure considered above. Binary trees and methods [3] have been chosen because we would like to develop a reusable PB that can be applied not only to the considered problem, but might also be adapted for high performance computational systems processing tasks (from a supplied input queue) with dynamically changeable priorities.

Note that reassignment of priorities might be done at any step, i.e. before and after entering a car to the garage. Thus, the originally supplied to a car sequence of instructions can be substituted after the car enters to the garage.

3. CENTRAL SUB-SYSTEM

The central sub-system (as well as the car sub-systems) is described using hierarchical specifications (hierarchical graph-schemes – HGS) [5,6] and models, such as hierarchical [5,6] and parallel [7] FSMs. All the project components have been synthesized from VHDL code using predefined templates for advanced FSMs. All the necessary operations, like count and shift, are executed directly in FSM processes and the proper synchronization is provided in a way very similar to [8].

Fig. 2 indicates four primary components of the central sub-system. Managing parking slots and priorities is the most complicated task. As we mentioned in the previous section, the PB implements a data structure representing binary tree and it is coded in memory as shown in Fig. 7 for the tree from Fig. 6.

Each cell at address *i*, *i*=1,2,…,28, represents slot *i* (see Fig. 4, 5) and contains four fields: priority value (written at the top of cells); address in the memory of the left sub-tree; address in the memory of the right sub-tree; and the state of the slot, such as «occupied» - O, «blocked or unavailable» - B, etc. (the first and the last fields are joined in Fig. 7 for simplicity). Number N indicates an absence of the relevant sub-tree (any unused for addresses value might be selected for such purposes). It is assumed that all garage slots have different priorities. If this constraint is not desirable then nodes with the same priority can be repeated in the tree.

Priority manager allows to change priorities of the slots and subsequently to rebuild the tree. It is done in three following steps: changing the priority value for the slot with the selected number *i*; removing the node *i*; and inserting the node *i* with the changed priority. The latter two steps are executed by hierarchical FSM implementing modules [3]. The first step is trivial.

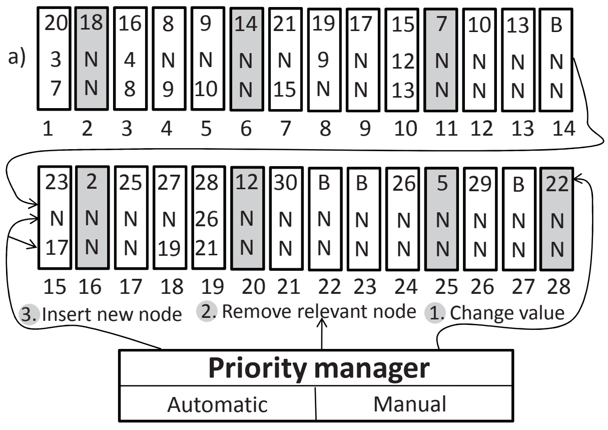


Fig. 7 Binary tree from Fig. 6 coded in memory.

Processing requests (see Fig. 2) are received from arriving and exiting cars. In the first case the following steps are carried out:

- The upper block in Fig. 2 verifies if there are free parking slots available and indicates the most preferable slot;
- The way to the slot from the entrance gate is computed and sent to the car sub-system;
- An instruction is sent to the bottom block in Fig. 2 and the entrance gate is opened;
- As soon as the car passes through the entrance, an instruction is sent to the bottom block in Fig. 2 and the entrance gate is closed;
- Location of the car is periodically (at the positions of sensors installed at crossroads) reported to the central sub-system and thus, the most preferable slot might dynamically be updated;
- As soon as the car is parked the data about the car ID and the occupied slot are reported to the central sub-system. This information is needed for retrieving the car when required.

Retrieving a car is requested by the car owner from outside of the garage through supplying the car ID. Then the following steps are carried out:

- The slot is determined through the car ID;
- The way from the slot to the exit gate is computed and sent to the car sub-system. Much like to the indicated above steps for parking cars the original way might be updated dynamically;
- As soon as the car approaches the exit gate the bottom block of Fig. 2 is informed and provides for

opening and closing the exit gate much like as it is done for the entrance gate;
- After passing the exit gate the car continues driving to a position where the owner is waiting for.

4. CAR SUB-SYSTEMS

Car sub-systems are responsible for driving cars from their current positions to the indicated destinations and for communications with the central sub-system. The control is based on typical scenarios such as that are demonstrated in Fig. 8:

- Direct motion (driving) between crossroads recognized by the signals from sensors S (a);
- Intelligent cruise control that permits to keep necessary distances between the car and a car in front (b);
- Parking to a slot (c, d);
- Turning (e);
- U-turn (f);
- Keeping distances (g);
- Exit from slots (h, i), etc.

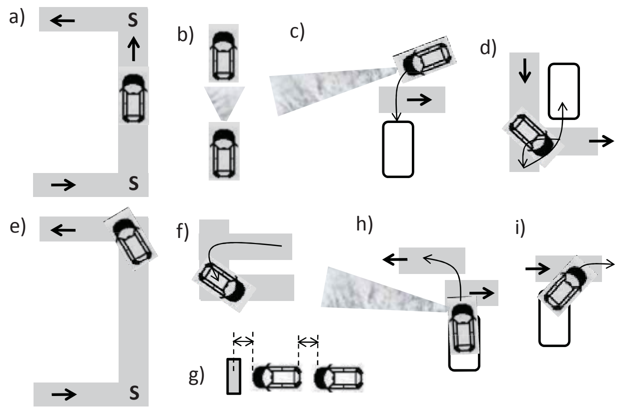


Fig. 8 Typical scenarios of car control.

The scenarios are implemented in an extendable set of predefined modules for hierarchical FSMs. Each module generates a sequence of steps needed for carrying out the relevant operation, i.e. it analyzes the signals from sensors and sends signals to actuators that set speed of the car, provide steering control, etc. As soon as the car passes crossroad sensors the relevant information is sent to the central sub-system indicating the exact location of the car.

5. IMPLEMENTATION DETAILS AND EXPERIMENTS

The considered automatic system for garage control was suggested as a work for M.Sc. thesis in 2008/2009 and has been completed successfully. The primary objective was to design all necessary electronic circuits including those that provide support for wireless interface, implement all the circuits in an FPGA and validate the intended functionality of the entire system (see Fig. 1) in a simulation environment (such as that was described in [9]).

Two modes of the system functionality are implemented. The first one permits to integrate all the hardware resources for the two sub-systems within the same FPGA (see Fig. 9). The second mode enables wireless (RF) interface between the two sub-systems implemented in different FPGA-based prototyping boards to be established.

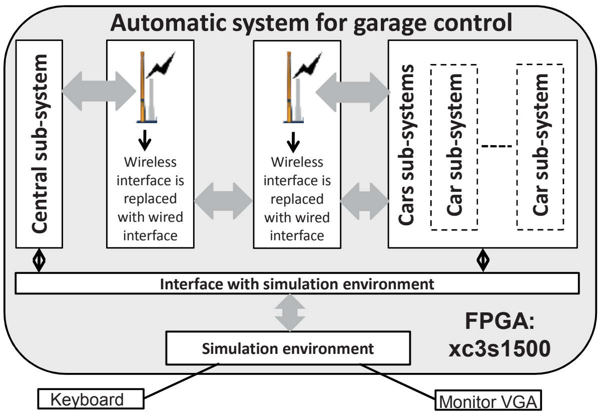

Automatic system for garage control
Central sub-system
Wireless interface is replaced with wired interface
Wireless interface is replaced with wired interface
Cars sub-systems
Car sub-system
Car sub-system
Interface with simulation environment
FPGA: xc3s1500
Simulation environment
Keyboard
Monitor VGA

Fig. 9 Simulation of FPGA-based circuits for the system.

Simulation environment displays projection of the garage to a VGA screen in a way shown in Fig. 10.

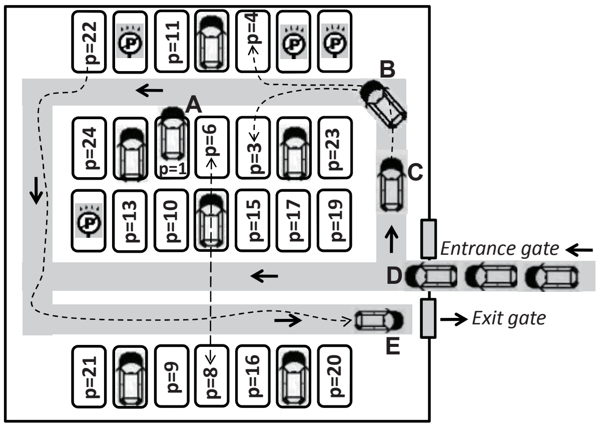


Fig. 10 An example of simulation.

Let us examine how the simulation can be used for different experiments. Positions and priorities of the slots in Fig. 10 are the same as in Fig. 4, a. Blocked slots are explicitly indicated by crossed symbols **P**. Allowed directions are shown by symbols →. Strings like p=10 indicate priorities. Suppose three cars A, B and C have already entered to the garage and the car D is entering through the opened entrance gate. The first car A is parking to the slot p=1 with the highest priority. According to the assigned priorities the subsequent cars B, C and D have to be parked in slots with the priorities p=3, p=4 and p=6 accordingly. However, such parking might create delay in the upper horizontal road. Thus, it is better reassign some priorities. For example, priorities p=6 and p=8 might be swapped, which would allow to park the car D to the middle bottom slot. If the priorities p=23 and p=4 could also be swapped then the car C could be parked to the first slot (marked with p=23) on the left-hand side. This would obviously avoid any delay. If the priority p=23 cannot be altered then another eventual solution would be to park the car B to the middle slot in the second line (original priority is p=6) and the car C to the slot with the priority p=3. Thus, different potential scenarios could be examined permitting to evaluate effectiveness of the designed system. For example, the car E is requested and retrieved from the slot p=22. The motion of this car (see dashed arrow line) is shown in dynamic on the screen. The simulation technique [9] enables to work with the models much similar to the physically implemented system, i.e. we can see all motions of the cars, opening and closing the gates, etc. Functionality can be adjusted by defining the following parameters from an attached keyboard (see Fig. 9):

- The number of arriving (new) cars per time slot;
- Generating a new car on the entrance by a user (by pressing a designated key on a keyboard);
- Speed of cars inside the garage;
- Requesting to retrieve cars by either users or signals from a random generator.

The complete automatic system has been designed (in Xilinx ISE from specification in VHDL), verified, implemented (in FPGA xc3s1500-4fg320) and tested in Celoxica RC10 prototyping board with a VGA monitor and a keyboard connected. The project can be adjusted (through generic VHDL statements) for different capacities of the garage and number of cars taking part in the simulation. At the moment not all the considered above capabilities of the automatic system have been implemented in hardware. For example, automatic changes of priorities (see Fig. 7) were modeled just in software, control of cars (see Fig. 8) does not deal with physical actuators and it is applied just to car models. Besides, this work does not provide a contribution in the scope of electrical interfaces with sensors and actuators.

The results of experiments and analysis can be summarized as follows:

- All basic circuits have been implemented and verified in hardware, which allows to conclude that from the logical point of view the intended functionality is correct;
- To validate methods, data structures and models a simulation in software based on a general-purpose language (namely C/C++) has been used. The results permit to conclude that such type of simulation is very helpful for eliminating potential errors at initial stages.
- Implementation of all basic circuits (see Fig. 1 and Fig. 9) including wireless communication has been done in FPGA and this confirms the correctness of the design;
- A number of improvements and experiments might be provided in future on the basis of the obtained results, such as comparison and analysis of different PB architectures and the relevant data structures, exploring alternative strategies for the priority manager, investigation of new useful scenarios, comparing processor-based and FSM-based implementations, etc.
- The visual tools that have been developed are very effective for experiments with alternative design techniques in an environment close to real world problems.

6. CONCLUSION

The paper describes functionality, structure, design and implementation in hardware (in FPGA) of an automatic system for garage control. The system is decomposed into two parts: a central sub-system providing top-level management of the garage; and a car sub-system providing signals to drive the car from pre-assigned positions to indicated destinations. The most important functional component of the central sub-system provides for management of parking slots and priorities. All topics relevant to this component were considered in detail. The implemented system was verified in a simulation environment, which permits to conclude that the desired functionality is correct. The results of simulation might be very helpful for exploration of advanced parking facilities, evaluation of critical situations and the required resources.

REFERENCES

[1] P.J. Ashenden, *Digital Design. An Embedded System Approach Using VHDL*, Morgan Kaufmann, 2008.

[2] D.A. Patterson, J.L. Hennessy, *Computer Organization and Design*, Elsevier, 2005.

[3] V. Sklyarov, I. Skliarova, “Modeling, Design and Implementation of a Priority Buffer for Embedded Systems”, Proc. 7th Asian Control Conference (ASCC09), Hong Kong, August, 2009.

[4] B.W. Kernighan, D.M. Ritchie, *The C Programming Language*, Prentice Hall, 1988.

[5] V. Sklyarov, *Synthesis of Finite State Machines Based on Matrix LSI*, Minsk, Science and Techniques, 1984.

[6] V. Sklyarov, “Hierarchical Finite-State Machines and Their Use for Digital Control”, IEEE Trans. on VLSI Systems, 1999, vol. 7, no. 2, pp. 222-228.

[7] V. Sklyarov, I. Skliarova, "Design and Implementation of Parallel Hierarchical Finite State Machines", Proc. 2nd Int. Conf. on Communications and Electronics – HUT-ICCE, Hoi An, Vietnam, June 2008, pp. 33-38.

[8] P.P. Chu, *FPGA prototyping by VHDL examples*, Jonh Willey & Sons, Inc, 2008.

[9] V. Sklyarov, I. Skliarova, "Virtual Environment for Prototyping and Experiments”, Proc. ICCAS-SICE, Fukuoka, Japan, August 2009.