A Control Design Approach for Controlling an Autonomous Vehicle with FPGAs

Anderson Pereira Correia, Carlos Humberto Llanos; Rodrigo Willians de Carvalho and Sadek A. Alfaro
Departamento de Engenharia Mecânica, UNB - Universidade de Brasília
Email: {Anderson, llanos, willians, sadek}@unb.br

Carla Koike
Departamento de Ciência da Computação, UNB - Universidade de Brasília
koike@unb.br

Edward David Moreno
DCOMP/UFS - Departamento de Ciência da Computação – Universidade Federal de Sergipe
edwdavid@gmail.com

***Abstract*— This paper describes the implementation of a platform based on reconfigurable architecture and on concepts of virtual instrumentation and its application to the hands-free driving problem. The novelty of this approach is the use of both reconfigurable systems (for developing the car's controller) and virtual instrumentation issues for developing a high-level abstraction testing and simulation environment. The implemented platform permits (a) to control directly the real vehicle using control commands that are sent using a keyboard and (b) to simulate the control process in a virtual environment, using a virtual instrumentation approach. The car control system was developed in a microcontroller with several peripheral embedded in a Field Programmable Gate Array (FPGA). The communication between the FPGA-based control system and the car is accomplished through an electronic module, which comprises several insulating and power circuit boards. The virtual instrumentation approach (for simulation and controller design objectives) was used for implementing a high-level abstraction simulation environment in LabVIEW tool, which allows representing the movement of the car in real time. The communication between the simulator and the controller is accomplished through a serial interface in which a RS-232 based protocol was implemented. The user can send commands to the control system through a keyboard with a PS2 interface. This approach opens a great variety of possibilities to validate and simulate solutions for several problems in robotic and mechatronic areas. The tests and initial overall system validation were accomplished in the simulator environment. Then, the simulation results were compared with the movement variables of the real car, which were gathered in real time. This approach makes possible to test and to validate the control system with low cost and more safety.**

***Index Terms*— Reconfigurable Computing, Embedded Processors, Virtual Instrumentation.**

I. INTRODUCTION

This paper describes a new design flow for the design of complex systems, which involves the application of reconfigurable devices (for implementing the electronic control modules) and virtual instrumentation issues.

The great complexity of the current systems, involving mechanical, electromechanical, electronic and computational parts, stimulates the introduction of new design methodologies. An important point is that the design methodologies must offer high abstraction level for verification/simulation tasks during the overall design. Several hardware manufacturers offer design tools, allowing verification/simulation of the electronics system on a high abstraction level. Similar methodologies can be applied for the design of mechanical or software modules of a complex system. The problem becomes difficult for testing, validating or verifying the whole system. In this case it is possible to use the virtual instrumentation approach in order to represent the overall system (or part of it) on a high abstraction level.

Virtual instrumentation tools (LabVIEW tool, for instance) allow to represent in real time several system's parts such as instruments, displacement, kinematics behavior of the objects (as robots or vehicles), mathematical operations over the signals (digital and/or analogical), among others. The signals coming from a controller can be routed to the virtual environment via data acquisition boards and, in the same way, the simulator/emulator can send electronic control signals in such a way to represent the current status of different virtual objects (such as electronic or mechanical parts, among others). Virtual instrumentation combines mainstream commercial technologies such as the PCs, with flexible software, and a wide variety of measurement and control hardware. Then, engineers and scientists can create user-defined systems, which meet their exact application needs.

The complexity-increasing evolution can be observed in different areas of the design of complex system. In the case of digital design area, the use of reconfigurable architectures allowed the implementation of more flexible systems based on FPGA (Field Programmable Gate Array) platforms. FPGAs provide an array of logical cells that can be configured to perform a given function by means of configuration bitstream. This bitstream is generated by a software tool, and it usually contains the configuration information for all components. An FPGA can have its behavior redefined in such a way that it can implement different digital systems on the same chip. Fine grain FPGAs allow the user to define a circuit at gate level, working with bit wide operators. This kind of architecture provides a lot of flexibility, but takes more

doi:10.4304/jcp.5.3.360-371

time to reconfigure than coarse grain reconfigurable platforms rDPAs (reconfigurable Data Path Arrays or arrays) of rDPUs (reconfigurable Data Path Units) [2]. In coarse grain reconfigurable platforms, the user does not provide details at gate level but specify the configuration in terms of word wide operations; in other words, a functional unit is configured to operate over n-bit data, and the configuration just specifies one among a set of available operations. The amount of configuration bits in this case is much less than in the fine grain FPGAs. Some FPGAs allow for performing PR (Partial Reconfiguration), such that a reduced bitstream reconfigures only a given subset of internal components. FPGA devices have been used for automation and control system rapid prototyping and they make possible to develop high performance systems in short periods of time. Besides, it is possible to have a smaller number of devices at reasonable costs.

Many current digital systems are based on the use of processors (based on the von Neumann Model), which are embedded in FPGAs, jointly with several hardware parts, described through HDL (Hardware Description Language) languages. The term SoC (System on Chip) [3] and [15] has been widely used in automation/control system design, communications systems, among others, which involves the use or implementation of different IPs (Intellectual Properties), and a full integration on the FPGA component. The SoC concept describes the fact that the whole functionality of the system is placed on a single chip. The structure of SoC devices with FPGA is fully flexible and can easily respond to changes in the control system logic. The capability of modifying the logic enables the control system to implement future additions with ease. Complex systems embedded into the FPGA, (eg. DSP, Soft/Hard processors among others), have been widely used in the industrial world.

We applied our reconfigurable system technique (used to implement the car controller) and virtual instrumentation based design methodology to the hands-free driving problem. Several research works are being developed on vehicle control design as well as CLMR (Car-Like Mobile Robot) [8], applying a variety of techniques based on complex mathematical models [10], neural networks [11] [13], genetic algorithms, fuzzy logic, to cite only a few. Steering a car is constrained by restrictions in the car's capability mechanism and the environment. Due to these reasons, it is very difficult to design a continuously global controller for a car in order to perform all the maneuvering behaviors. Over the years, numerous systems have been developed to provide automatic control for the hands-free driving problem of automobiles [6]. These systems automate either steering control (related to as lateral control), throttle and/or brake control (related to longitudinal control), and the clutch control. When the automobile control involves all partial control systems, it is called AHS (Automated Highway System) [12].

Some researchers have reported the use of FPGA and LabVIEW applied to the either CLMR or hands-free driving problems. A FPGA implementation of a FGPC (Fuzzy Garage Parking Control) is discussed in [12]. The use of FPGA and LabVIEW is discussed in [9] for an accelerator control system design. It is clear that the solution for this kind of problems is far from straightforward and requires multidisciplinary knowledge of the fields of control, automation, robotics and hardware and software design (see section 2). In the work of Petko [11] and Zhao [16], several partial results about a controller design (based on reconfigurable architectures) and a simulator (implemented on LabVIEW) were presented. In this approach the developed simulator environment (based in virtual instrumentation) is used for simulation/verification tasks of the control system. Once the system is validated, the FPGA embedded control system can directly operate over the real vehicle.

In section 2 the related work about vehicular automation are discussed. In section 3, the overall architecture of the system proposed here is described, it presents basic concepts of the proposed embedded architectural system in the FPGA and discusses the defined command set for the control system. Section 4 describes the virtual environment for simulating the vehicle motion. Section 5 describes the communication protocol, which was defined for the simulation system environment. Before concluding, sections 6 and 7 describe our results and conclusions.

II. RELATED WORKS IN VEHICULAR AUTOMATION

There are two main tendencies in the vehicular automation area. In the first case, several vehicle automation systems have been proposed, which involves several developments allowing automatic vehicle maneuvers using both sensor systems and/or the trajectory planning applied for solving the vehicle automatic parking problem, among others. In the second case, several systems have been proposed and implemented for developing driver assistance systems for helping the driver for parking maneuvering and/or roadway guidance.

In the work of Gu and Hu [7], a vehicular automatic parallel parking system was adapted for installation either as a factory option or as a retrofit kit. A microprocessor-based controller exercises control over a hydraulic system for controlling vehicle's power steering cylinder. The system only controls the steering wheel and does not act over the clutch, brake, throttle and gear systems. This work does not report information about HMI (Human-Machine Interface) for monitoring the parking process. For parking task, sensors automatically provide inputs representative of the transverse and longitudinal relationships between the driven vehicle and the parked vehicle. The developed PSS (Parking Sensor System) is based on ultrasonic sensors.

In the work of Shimazaqui [24] a parking assistance system is shown, providing a driver with guidance on a driving operation in lateral parking. The system provides a camera mounted for monitoring a backward movement and a HMI for driving assistance during lateral parking

maneuver. Otherwise, the approach does not implement a real control system for steering wheel, clutch, throttle and gear systems.

In the work of Tanaka et al. [25] a parking assistant system is reported that comprises a camera for capturing the rear view, a steering wheel position/speed sensors for monitoring the current status and a controller, which is implemented via a embedded computer in the vehicle. The control system acts over the steering wheel, brake and throttle systems but the control over the clutch is not reported.

In the work of [17] the concepts of AVC (Automatic Vehicle Control) are applied in the implementation of the PATH system (Program on Advanced Technology for Highway). In this approach a control program was developed for both lateral and longitudinal (spacing and speed) control and it is only applied in automated roadways. Lateral control maintains the vehicle in the center of the lane (lane-keeping maneuver) and steers the vehicle to an adjacent lane (lane-change maneuver), while maintaining good passenger comfort at all times. The lateral control work is focused on the concept of cooperation between the vehicle and the roadway, with an intelligent vehicle receiving much of the information it needs from special elements installed in the roadway (for instance, magnetic markers embedded under the roadway for lateral guidance). Longitudinal control involves regulating the vehicle speed to keep proper spacing between vehicles. Then, a longitudinal spacing control system is developed, which involves sensors and mathematical models for the power-train (including internal combustion engine dynamics and tire/road frictional interface).

In the work of Wada et al. [19] a driver assistance system to aid in the vehicle parking process is proposed, which maintains the driver in the vehicle control loop. The system involves sensor devices and path planning algorithms for driver assistance issues, which also provides a HMI (Human-Machine Interface) for guiding the driver during the parking process in predefined parking bay.

In the work of Han et al. [18] a reliable automated steering control system was developed for the PATH system, which attends several requirements. The control system requires road markers or any other kind of road indicators in order to define the road. Additionally, the implemented system requires sensors in the car, recognizing the road markers and intelligence in the controller for driving the steering actuators, which steer the wheels.

The different related solutions can involve several levels of the automation issues. For example, neither the works of Wada et al. [19] nor [24] involve a development approach with regards to the automation of the steering wheel, the clutch, the brake or the throttle. Otherwise, the work of [7] only implements a hydraulic controller of the steering wheel. In contrast, the works of [19], [24] and [25] implement the HMI for the systems. In all works cited above, the control systems were developed for working with internal combustion engines except the work of [19], which can be adapted for other kind of engines.

Recently, Amudha et al. [21] discuss the use of FPGAs and microcontrollers embedded in FPGAs for different applications and focuses on soft core processors. Murakami [22] designed the redundancy controller in hardware for humanoid robot applications. Han et al. [23] has been implemented in a DSP-based embedded system to recognize five facial expressions on-line in real time for Robotic Emotion Recognition.

Finally, all the systems use an embedded computer in the car for implementing the control system for their applications, without exploring reconfigurable architecture techniques. Additionally, none of the works report the development of environments for simulation and/or verification issues.

III. THE PROPOSED EMBEDDED ARCHITECTURE

The overall control system is composed of an embedded control system based on the soft-embedded-processor Microblaze [14], which is implemented in a Spartan 3–based FPGA, and a virtual simulator environment implemented in LabVIEW. The architecture is shown in Figure 1, where a communication system is implemented using RS232 standard. Additionally, a keyboard is used for sending pre-defined commands to the control implemented in the FPGA.

The embedded microprocessor implements the main control of car tasks in software functions, namely: brake, clutch, steering wheel, gear and throttle sub-systems of a real vehicle. Each function was written in C language in a structured software approach. Several hardware modules were incorporated to the hardware design during the project specification: RS232 interface, buttons, display using the EDK tool (Embedded Design Kit) options [5], and so on. Finally, a specific keyboard module described in the hardware description language VHDL was added to the controller design.

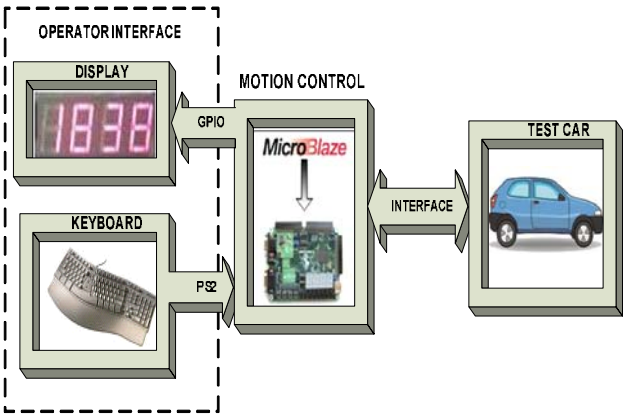


Figure 1. The Overall System Architecture

A simulator environment was developed in the LabVIEW system and it is connected to the controller through a RS-232 based interface. Additionally, a communication protocol was defined to achieve the

communication between the controller and the simulator. In this case, both simulator and the controller exchange information using a predefined data-package format. Figure 1 shows this approach, where the motion controller can be accessed using the operator interface, which includes a keyboard for editing control commands and a display for monitoring the status variables. The controller can interface with both the real car and the simulator tool that was implemented in LabVIEW.

The control module was defined using the EDK tool [5], in which the Microblaze processor is the system core. This processor has a RISC architecture with 32-bit general purpose registers, an ALU (Arithmetic Logic Unit), a shift unit, interrupts, among other possible peripherals. The EDK tool is an embedded development environment that includes a library of peripheral IP cores, where the Xilinx Platform Studio tool is employed for intuitive hardware system creation. Additionally, a Built-On Eclipse software development environment, GNU compiler and a debugger are also included. Figure 2 depicts the system, which includes the embedded system in the FPGA, the user interface and the communication with the simulator environment.

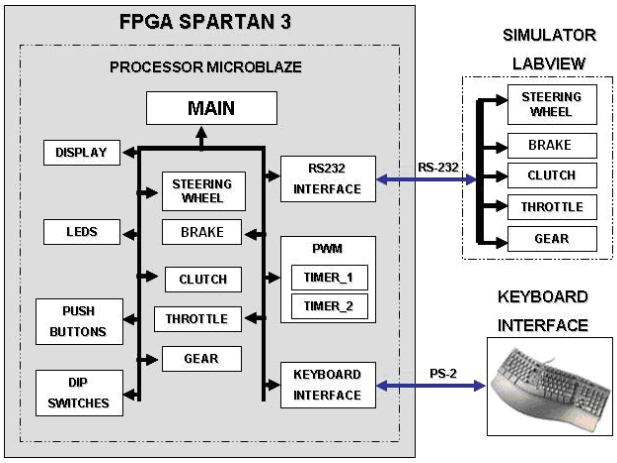


Figure 2. The embedded system and its interface

A. The hardware System of the Controller

The architecture of the embedded motion controller system is shown in Figure 3, which was designed and synthesized using the EDK. The communication of the processor with peripheral devices is achieved by the OPB bus (On-chip Peripheral Bus). There are several hardware peripherals related to the FPGA-based board resources such as display, keyboard, RS232, push-buttons, dip-switches and leds. The processor controls the operation flow of the system by running different special designed software functions, which were written in C language and stored in the bRAM-block (see Figure 3). The leds, display and pushbutton modules were automatically generated by the EDK system. On the other hand, the keyboard module was first described in a VHDL (Very High speed integrated circuit Hardware Description Language) file implementing the PS2 protocol and then included as a peripheral device in the overall design. The PWM blocks are responsible for generating modulated speed control signals of the DC-motors related to the throttle and gear devices. The PWM signals were implemented using Microblaze's timers, which can be added to the design according to the system needs. In this case, only two PWM modules have been generated; further details can be found in [20].

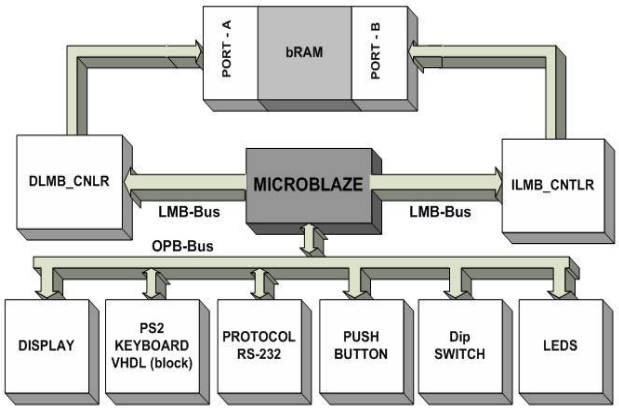


Figure 3. The Motion Controller System on the Spartan 3 based board

B. The Software Modules of the Controller

Once the processor system was configured and your peripherals were defined, all programming was made in standard C language, compiled and tested inside of the EDK environment. The module descriptions are the following (see figure 4):

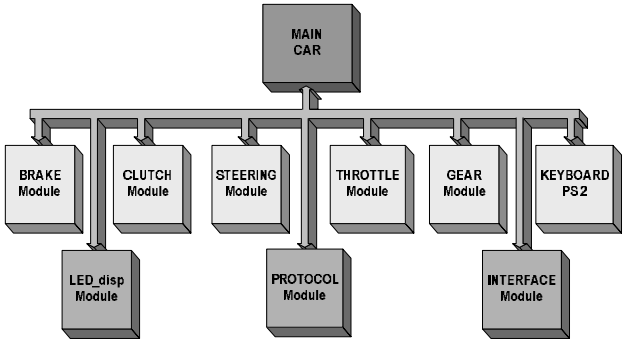


Figure 4: Software Project of the Control

a) The *brake.c* module: it receives a defined command to operate the car-brake (see section 3.C). The module verifies the current position of the brake and it gives the proper direction to the actuator. A PWM (Pulse-Width Modulation) signal is used to control the actuator-speed.

b) The *clutch.c* module: it receives commands from the user (see section 4) and verifies the current position of the clutch, executing a special procedure to drive the pneumatic-system. This module has an alternative way to execute the clutch control by a stepper-motor.

c) The *steering.c* wheel module: it receives defined commands (see section 3.C) to achieve a user-defined position. The module verifies the current

position and it gives the proper direction to the wheel actuator.

d) The *throttle.c* module: It works in two stages: the first one works for controlling the butterfly valve position, which is measured by a potentiometer. The second one executes a control strategy, where a rotation reference is set by the user. Then, the system controls the position until the required rotation is accomplished. A PWM signal is used to control the actuator-speed.

e) The *gear.c* module: this module receives the command of the operator (see section 4) and verifies the current position for changing the gear-position. This is achieved by two DC-motors, which move the gear-lever in the X and Y axis in a predefined way. The DC-motor's speed is controlled by two PWM-signals.

Fig. 5 shows the C code of the gear module (the gear function). The gear position is defined in the x, y axis, which represent columns and rows respectively. The definition of the new-gear position is defined in lines 3 to 21 (see Figure 5), where the first, second, third, fourth, neutral and reverse gears are identified.

```
1   void gear(Xuint8 gear) {
2   current_columm = 0;
3   if(pos_x_gear == POS1X)
      { // init the gear-position identification process
4     if(pos_y_gear == POS3Y) new_gear = 1;
      // the first gear
5     else if(pos_y_gear == POS1Y) new_gear = 2;
6     else new_gear = 'U'; // unknown gear
7     current_columm = 1;
8   }
9   else if(pos_x_gear == POS2X) {
10    current_columm = 2;
11    if(pos_y_gear == POS3Y) new_gear = 3;
12    else if(pos_y_gear == POS1Y) new_gear = 4;
13    else if(pos_y_gear == POS2Y) new_gear = 'N';
    // the neutral gear
14  }
15  else if(pos_x_gear == POS3X) {
16    current_columm = 3;
17    if(pos_y_gear == POS3Y) new_gear = 5;
18    else if(pos_y_gear == POS1Y) new_gear = 'R';
19    else new_gear = 'U';
20  }
21  else new_gear = 'U';
      // unknown gear
22  if(gear == new_gear) {
      // the new position was achieved
23          adjust_pwm_gear('P');
24          • • •
25          • • •
26          • • •
27  }
28  // First_gear Treatment
29  else if(gear== 1) {
      // to accomplish the new position
30    if((current_columm == 1 && pos_y_gear < POS3Y)
            ||
        (current_columm>1 && pos_y_gear< POS2Y)) {
31    if(current_columm == 1)
            dif = POS3Y – pos_y_gear;
32    else dif = POS2Y – pos_y_gear;
33          adjust_pwm_gear('Y');
34                  • • •
35        }
36  • • •
37  }
```

Figure 5. The C code of the Gear Module

The state variables pos_x_gear and pos_y_gear are used for representing the current gear-position. In the real car the positions are measured by means of two potentiometers (for x and y axis). The potentiometer sensors (adapted in the real car) send values, which are compared with predefined constants, namely POS1X, POS2X, POS3X, POS1Y, POS2Y and POS3Y. For instance, lines 3 to 8 identify if the current gear is the first-gear. The other gears are identified in lines 9 to 21 (including the reverse and unknown gears). When the new gear position is defined the system starts the process in order to achieve the new gear position. Lines 28 to 35 show the process to accomplish the first gear where the adjust_pwm_gear function is called with parameter "Y" for controlling the corresponding DC motor, sending an appropriated PWM signal (see line 33). The same program also tests if the current position is equal to the desired one, which is indicated by the new_gear variable (see line 22). In this case the adjust_pwm_gear function is called for stopping the DC motor movement, using the "P" parameter (see line 23).

C. The Commands for Interface Control System

Several commands were defined in order to control the car and their definitions, whose specific syntax and semantics are described in table I and II. The commands are organized into two sets, describing both automatic and manual modes (see tables I and II, respectively).

The first mode defines commands for debugging actions, including arrow keys for increasing/reducing the current positing of steering wheel, clutch and engine rotation, among others. The second mode defines commands for using either via keyboard or into a C program. In this case each command was implemented in a dedicated C function, using two parameters (*x* and *y*, see table I). For example, the command related to the clutch (EBxy) can have the parameter *x* defined as *A*, *B* and *C*, indicating three different semantics: press the clutch, fast disable of the clutch, and disable the clutch until *y* % of the final position, respectively (see table I). The parameter *y* is only valid for the third case (EBCy, see table I), where the *y* parameter represents the final position that the clutch will reach.

The commands are sent by the user using the keyboard and then the Microblaze identifies and processes them, before sending the appropriate control signals (to the actuators) using the RS232-base protocol for the simulator environment. For the real car the signals are directly sent in parallel, using the expander connectors of the FPGA-based board.

IV. THE SIMULATOR ENVIRONMENT

The main task of the simulator environment is to simulate the kinematics and general behavior of the vehicle in normal situations. The concepts of Virtual Instrumentation, by programming in LabVIEW environment, were applied in order to generate the

appropriated signals, according to the control and status variables.

TABLE I

COMMAND-LINES FOR THE CONTROLLER

Syntax	Parameter 1 (*x*)	Parameter 2 (*y*)	Semantics
DI*xy*	*D*, *E* or *C*	integer of 0 to 60	Turn the front-wheels right in *x* degrees (*x* = *D*), or turn the front-wheels left in *x* degrees (*x* = *E*), or align the front-wheels (*x* = *C*).
FRE*xy*	0 or 1	-	FRE‘1’: press the brake or FRE‘0’ – release the brake
AC*xy*	*B* or *R*	0 - 100 If x = *b*. 0 – 9999 If x = *r*	AC*By*: put the throttle butterfly valve at *y*% of the maximum position. AC*Ry*: activate the throttle butterfly valve until the rotation reaches a *y* value
CAB*xy*	0 to 6	-	CAB*x*: put the gearshift at *x* position.
EB*xy*	*A*, *R* or *C*	0 – 100 if *x* = *C*	EB*A*: press the clutch. EB*R*: execute a fast disable of the clutch. EB*Cy*: disable the clutch until *y* % of the final position.
C*xy* or F*xy*	*A*, *C* or *F*	0 - 100 for *Cx* 1 – 9999 for *Fx*	C*xy*: modify PWM duty cycle at *y*% for: a) the brake (*x* = *F*), b) the throttle valve (*x* = *A*) and c) the gear (*x* = *C*). F*xy*: modify the PWM frequency for: a) the brake (*x* = *F*). b) the throttle valve (*x* =*A*). c) the gear (*x* = *C*). The frequency is modified for *y* KHz (LabVIEW or the real car).
P*xy*	*A* or *R*	positive integer	P*Ay*: rotate the clutch steeper motor actuator *y* steeps in the up direction. P*Ry*: rotate the clutch steeper motor actuator *y* steeps in the reverse direction.

The general structure of the simulator environment system is depicted in Figure 6. The architecture is defined by the global-variable-module, the serial-module, the signal-variable-module and the trajectory-module (see Figure 6).

TABLE II

MANUAL COMMANDS

Syntax	Semantics
↑	Increase the engine rotation
↓	Reduces the engine rotation
←	Rotate the front-wheels to the left
→	Rotate the front-wheels to the right
W	Move the gearshift to the up side
S	Move the gearshift to the down side
A	Turn the gearshift to the left
D	Turn the gearshift to the right
Space	press/release the brake
E	press the clutch
Q	Fast release of the clutch
1	Put the first gear position
N	Put the neutral gear position
R	Put the reverse gear position

The global-variable-module allows the communication among the all other modules, representing the status variables in 8-bits words. The serial-module performs the communication though the serial based protocol, allowing the user to configure several parameter such as bits per character, start/stop bits, parity and bit rates for transmission. Data are transmitted in a RS-232 interface and the receiving bytes are separated and interpreted by the module following the defined protocol (see sec. V).

The communication process executes four steps in sequence: open-serial-port, receiving-data, separating-bytes and close-serial-port (Figure 7 shows these steps). Whenever the close-serial-port steep is executed the module executes the open-serial-port steep again.

$$\dot{x} = v.\cos(\theta).\cos(\phi) \quad (1)$$

$$\dot{y} = v.\sin(\theta).\cos(\phi) \quad (2)$$

$$\dot{\theta} = v.\frac{\sin\phi}{l} \quad (3)$$

Where:

- ϕ is the angle of the wheels;
- θ is the vehicle angle with regard to the *X* axis;
- is the distance between the vehicle center to the *Y* axis
- *y* is the distance between the vehicle center to the X axis;
- *l* the distance between the vehicle’s wheels;
- *v* is the vehicle speed.

Some parts of the trajectory-module were directly implemented in C language in order to represent the canonical equations.

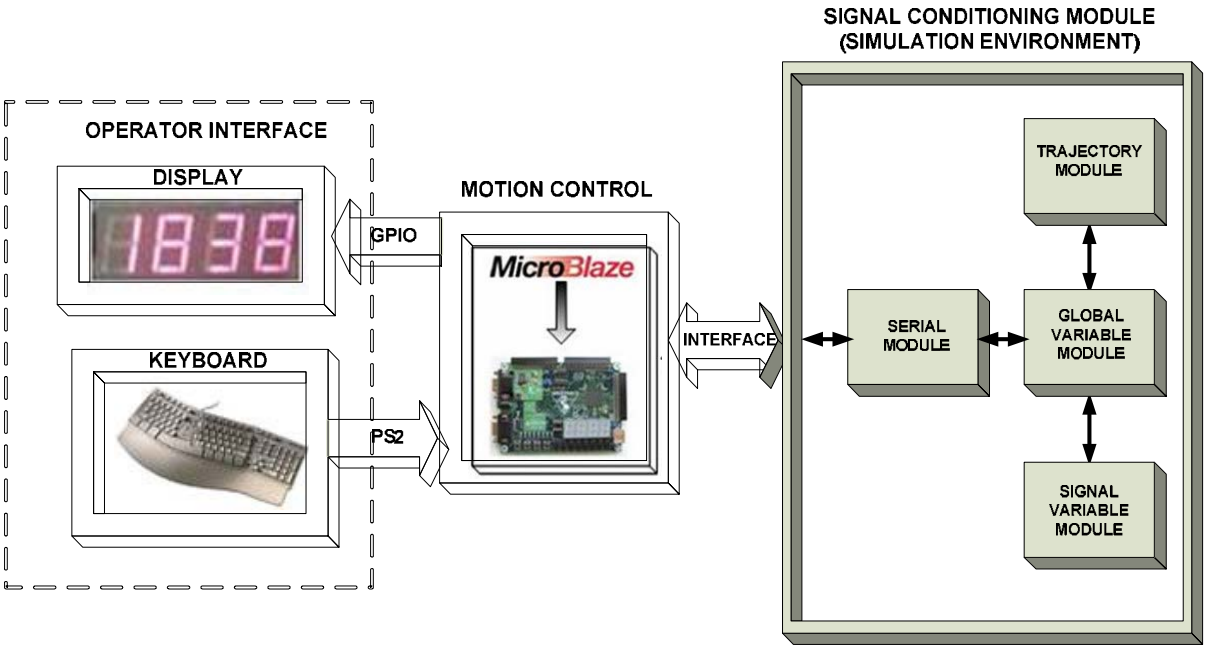


Figure 6. The general architecture of the simulator environment

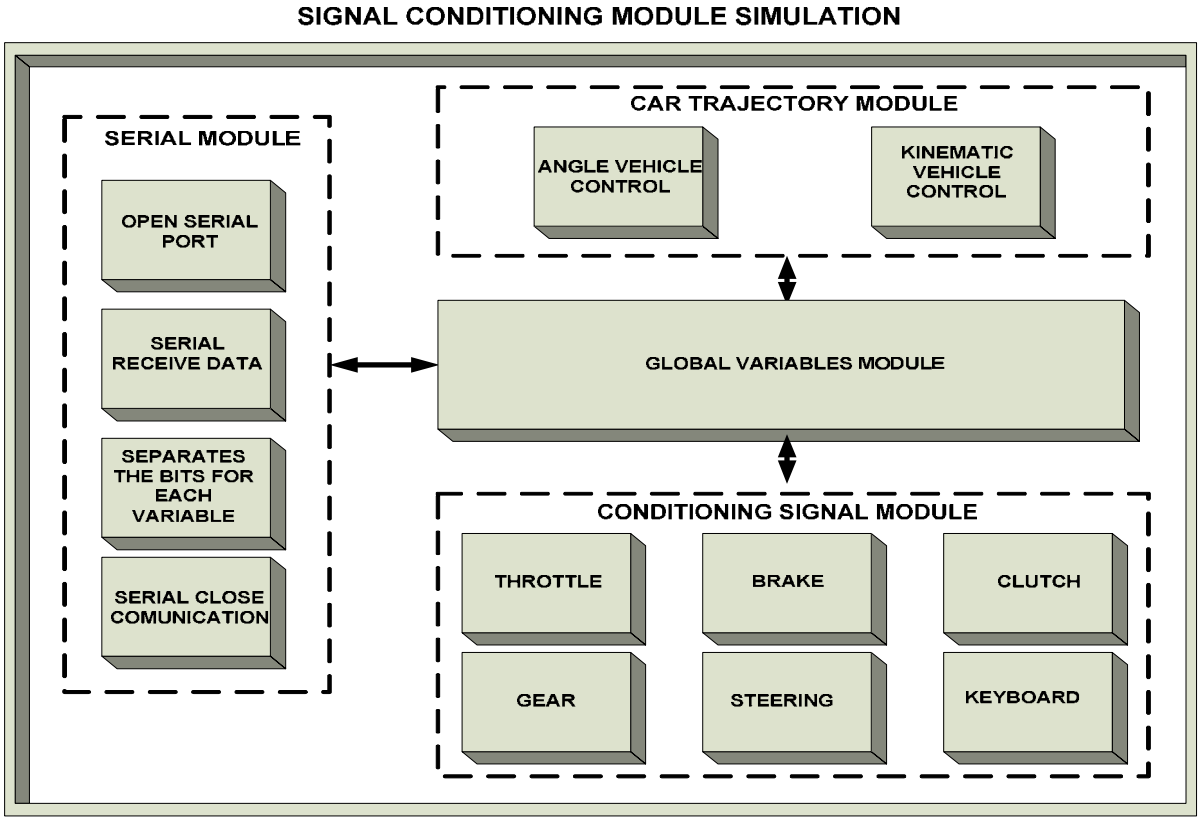


Figure 7. The internal structure of the simulator

Fig. 9 shows the LabVIEW sub-module of the throttle control in the virtual car, where a C code was embedded in the LabVIEW VI (Virtual Instrumentation).

The loops in each module are executed with a rate of 1KHz, allowing the user to define speed parameter for the virtual actuators. For instance, if the user needs to increment the throttle variable in 50 units per second then the variable is increasing at 0.05 units per execution cycle. The user's interface of the simulator environment is shown in Figure 10 and it represents the car position and several blocks for monitoring the current engine rotation, gear position, throttle, among others.

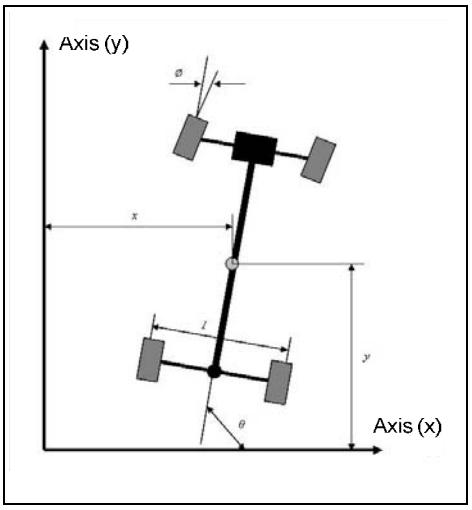


Figure 8. The Kinematics Variables using the three canonical equations

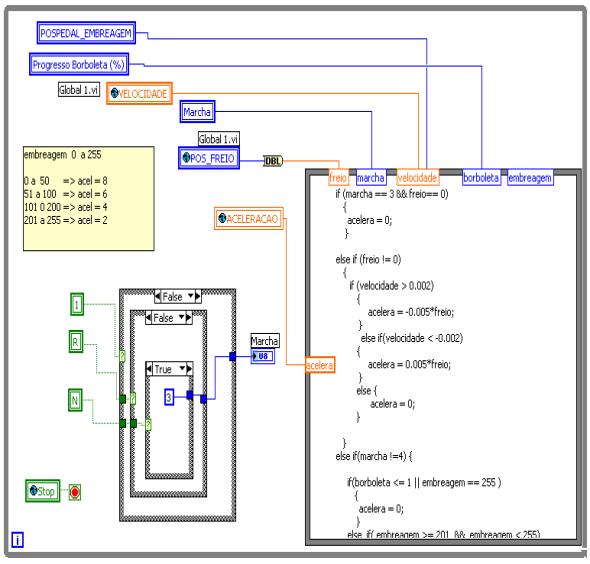


Figure 9. Software Structure of the Virtual Simulator Environment – The Throttle Control

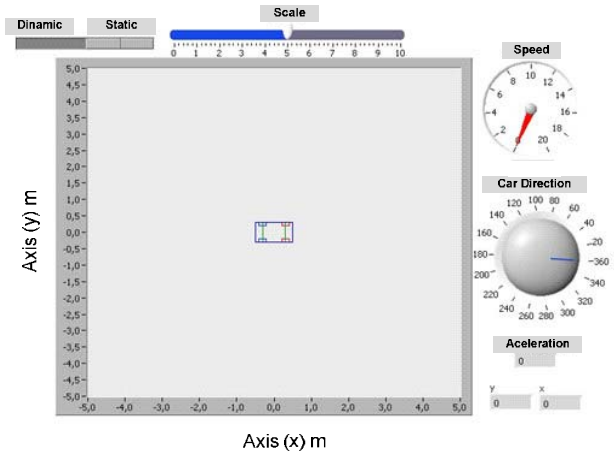


Figure 10. The user interface in the LabVIEW

V. THE COMMUNICATION PROTOCOL FOR THE SIMULATOR SYSTEM

A communication protocol was defined in order to implement a full-duplex communication between the control module (FPGA) and the LabVIEW program. The controller sends to the LabVIEW a 3-bytes package, where the first one represents a specific car state-variable, encoded in 4 bits (namely, front-wheel_1, front-wheel_2, x-position of the gear, y-position of the gear, brake-position and clutch-position) (see Figure 11). The second byte represents the information for controlling the clutch, in which the 4-most-significant bits are used for the stepper-motor signals and the other bits for electro-valve system control.

The last byte is used for generating and sending PWM signals for the throttle (2-bits), steering wheel (2-bits) and brake (2-bits). In this case, the first bit of the throttle is used to represent the direction and the second is used for generate the properly PWM signal. Given that the packages are sent in a sequence, the simulation environment program is capable to rebuild the PWM signal using only one bit in the serial communication.

On the other hand, the information coming from the simulation program (the feedback of the state variables) is encoded using two bytes (see Figure 12). The first byte is used to encode a specific state variable (front-wheel_1, front-wheel_2, x-position of the gear, y-position of the gear, brake-position, and clutch-position), using the 3-most significant bits. The second byte represents the current value of the corresponding state variable (for example, the clutch variable). In this case, the program responds to the controller about the required state information.

VI. RESULTS FROM SIMULATION AND REAL TESTS

The results obtained with the implementation of the system were distributed in four topics: FPGA synthesis results, testing car results, simulator environment results and simulator-results vs. real-car-results.

A. The FPGA Synthesis Results

The FPGA synthesis results were obtained in the EDK project report [14]. The results are shown in Tab. 3 for the main modules of the control system, where a Spartan 3 device (xc3s200ft256-4) was employed for the hardware implementation of the controller.

The main resources consumption is related to the Microblaze implementation. The clock frequency is shown for each implemented device. The results (in percentage of the total of resources available in Spartan 3 device) are related to slices, slices-flip-flops, LUTs, IOB, Ram-Blocks (Bram).

There are also timing results for each hardware modules (for instance, microprocessor and PWM modules).

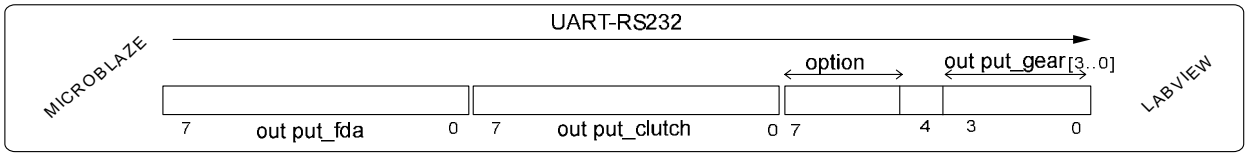


Figure 11. The protocol structure to send data from Microblaze to the simulator

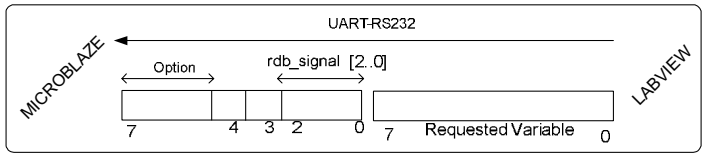


Figure 12. The protocol structure to send data from the simulator to the Microblaze

In this case, the critical frequency (the lowest operation frequency) is for the 7-segment driver (about 68 MHz, see line 6, column 7), representing the global timing constraint for the overall system. Given that the used FPGA based board works at 50 MHz, this critical frequency has not impact in the control device and does not represents a bottleneck in the overall control system performance. Table III depicts only a peripheral implementation for one PWM signal, but additional PWM devices can be easily added in the design depending on the requirements.

TABLE III

SYNTHESIS RESULTS IN THE EDK TOOL

Module	Slices (%)	Slices flip-flops (%)	LUTs (%)	IOB	BRAM (%)	Max Freq (MHz)
Microblaze	43	14	29	751	0	91.128
BRAM-block	0	0	0	119	66	203.707
DIP-Switches – 8 bits	2	1	0	119	0	135.612
Push_Buttons – 3 bits	2	1	2	64	0	138.927
Interface_I/O	10	8	2	285	0	134.953
Opb_7segled_0	9	4	5	69	0	68.362
Ps2_Keyboard_0	2	1	1	64	0	100.120
PWM_I/O	2	1	0	98	0	138.658
PWM_timer_0	13	8	7	67	0	98.348

B. The Simulation Environment Results

The simulator environment results are shown in Figure 13 as a sequence of illustrations demonstrating the vehicle movement controlled by the same commands shown in tables I and II. Figures 13.a to 13.h show the car in the simulation environment, moving through the window area. A command sequence was sent through the keyboard in order to simulate a parking maneuver. Figure 13.h shows the final position of the car. All the commands were executed in the manual-mode set (see table II).

C. The Testing Car Results

Tests of car movement control were accomplished with the objective of validating the movement controller and evaluated the feasibility of the system. The vehicle was controlled through commands sent through the keyboard. Figure 14 shows a sequence of images of the film of the car movement test. In Figure 14.a the car is stopped whereas Figure 14.b shows the car beginning the movement (the processes to accomplish the first-gear was already finished). Several maneuvers are shown in Figures 14.c to 14.h, which imply the use of the clutch, the steering wheel and the brake. All the commands were sent to the FPGA embedded controller through the keyboard.

D. Simulator-results/Real-car-result (comparing the results)

Fig. 15 allows the performance comparison between the real car and the simulator environment for a specific case. Figure 15.a shows the position signal of the throttle transducer (using a mathematical model of a real transducer), obtained in the simulator environment.
The throttle position was changed through the time and the maximal acceleration was obtained between 100 and 125 seconds after the throttle position change. On the other hand, Figure 15.b shows the real signal coming from the throttle transducer (the real throttle potentiometer). Notice that acceleration and deceleration curves are very similar. Both signal amplitudes (from the simulator and the car) were obtained using 8-bits resolution (the value of maximal acceleration signal is 255).

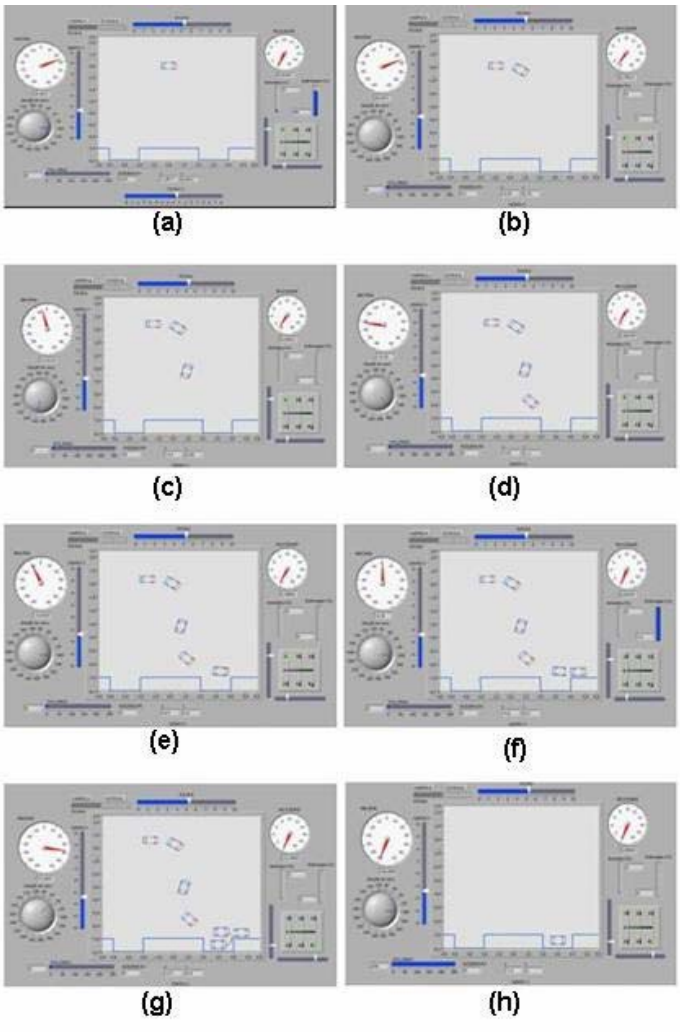


Figure 13. The Simulator Environment Results

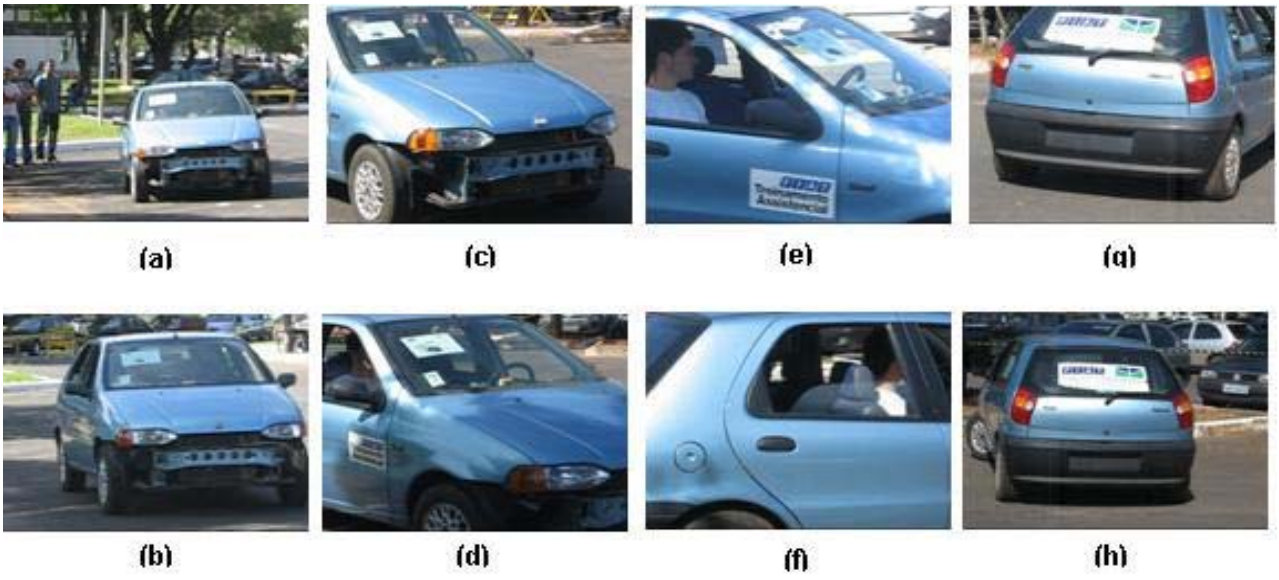
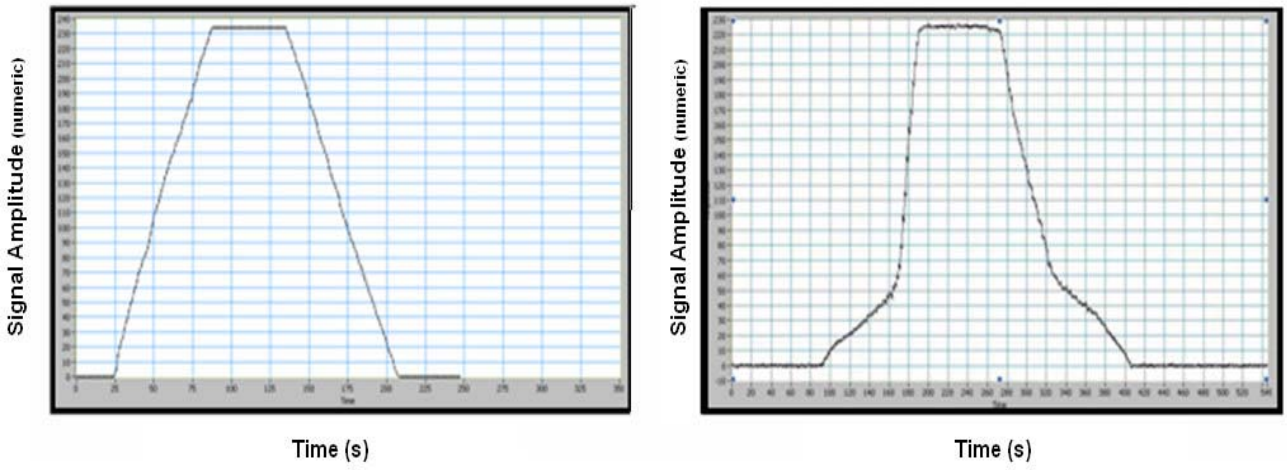


Figure 14. Results of Car Test



(a) The throttle signal in the simulator (b)The throttle signal in the car

Figure 15. Comparing throttle behavior of real car and the simulated car

VII. CONCLUSIONS

A completed car control system was implemented using a FPGA-embedded processor. The implemented car motion system comprises the steering wheel, clutch, gear, brake and throttle subsystems. To accomplish the controller's implementation, specific commands (whose syntax and semantics were defined in this work) allow the maneuvering of the car. The commands can be also used into the control program running in the embedded microcontroller. That is, the commands are really specific functions that are implemented in C language and allow the implementation of the automation strategies of the car. This approach permits to use the commands in order to follow a previously defined trajectory, which is fundamental for automatic parking maneuvering issues and to solve the hands-free driving problem. Regarding HMI, the implemented controller provides monitoring facilities for the different status variables (for steering-wheel, the throttle, the clutch, the brake and the gear) in both cases: the real car and the simulator in a similar way as reported in [19] and [25]. One important and original contribution of this work is that it implements the whole automation of a normal car, even with a mechanical gear. The results of this research can be easily applied in the case of people with disabilities, which need to drive adapted cars.

The simulator allowed the overall system validation in a high abstraction level, very close to the real situations. Virtual instrumentation based in LabVIEW is a powerful tool for representing the reality, and to the best of our knowledge, no similar simulation system can be found in the literature that allows the validation of a complex control system on this abstraction level.

Additionally, a RS-232 based protocol was defined and tested to allow the user to send the commands to the controller (by typing in a keyboard). The same protocol allows the controller sends predefined data packages to the LabVIEW environment in order to update the current status of the virtual car in real time.

The tests in the simulator and in the real car have shown the suitability of this design flow for validating complex mechatronic designs with a high reliability and safety. Several low level security strategies can be added to the current system for avoiding accident (e.g. collisions and critical situations) and these strategies can be also introduced into the simulator environment.

Otherwise, FPGAs are very suitable devices for implementing several automation and control techniques due to the fact that allow for embedding both typical microprocessor such as ARM family [1] and DSPs. In the last case, DSPs allow the implementation of specific algorithm for digital signal processing using several embedded resources such as multipliers and adders. Hence, FPGA approach opens a wide variety of possibilities for implementing, validating and simulating solutions for several problems in the robotic and mechatronic areas.

REFERENCES

[1] Altera, 2009. Available at http://www.altera.com. Accessed in May 2009.

[2] Becker, J. and Hartenstein, R., 2003, "Configware and Morphware going mainstream". J. Sys. Arch. 49: pp.127-142.

[3] Donecker, S. M., Lasky, T. A., Ravani, B., 2003, "A Mechatronic Sensing System for Vehicle Guidance and Control". IEEE-Transactions on Mechatronics, Vol.8, n.4, December, pp. 500 – 510

[4] Dudek, G. and Jenkin, M., 2000, "Computational Principles of Mobile Robotics". Cambridge University Press, Cambridge, UK.

[5] EDK, 2009, "Platform Studio, User Guide". Available at www.xilinx.com/ise/embedded/edk_docs.htm. Accessed in April 2009.

[6] Giove, D., Martinis C. D., Mauri, M., 2004, "Reconfigurable Hardware Resource in Accelerator Control System". EPAC, Lucerne, Switzerland, pp. 701 – 703.

[7] Gu, D., Hu. H., 2002, "Neural Predictive Control for a Car-like Mobile Robot. International Journal of Robotics and Autonomous Systems", Vol. 39, No. 2-3, May, pp. 1–15.

[8] Li, J. H, Lee, Li, P. M., 2005, "A Neural Network Adaptive Controller Design for Free-Pitch-Angle Diving Behavior of an Autonomous Underwater Vehicle". Robotics and Autonomous Systems. Elsevier, 52, pp. 132 – 147.

[9] National Instruments, 2009. Available at http://www.ni.com./labview/whatis/. Accessed in May 2009.

[10] Paromtchik I. E., Laugier C., Gusev. S. V., Sekhavat S., 1998, "Motion Control for Autonomous Car Maneuvering". Available at http://citeseer.ist.psu.edu/184744.html. Accessed in April 2009.

[11] Petko, M., Uhl, T., 2001, "Embedded controller design-mechatronic approach". IEEE, Second Workshop on Robot Motion and Control, pp. 195 – 200.

[12] Tan, H.S., Guldner, J., Patwardhan, S., Chen, C., Bougler, B., 1999, "Development of an Automated Steering Vehicle Based on Roadway Magnets A Case Study of Mechatronic System Design". IEEE/ASME Transactions on Mechatronics, Vol. 4, No. 3. pp. 258 – 271.

[13] Tzuu-Hseng, S., Chang, S-J., Chen, Y-X., 2003, "Implementation of Autonomous Fuzzy Garage-Parking Control by an FPGA-Based Car-Like Mobile Robot Using Infrared Sensors". International Conference on Robotics & Automation, Taipei, Taiwan, September, pp. 3776 – 3781

[14] Xilinx. Inc, 2009. Available at http://www.xilinx.com/. Accessed in May 2009.

[15] Yang, E., Gu, D., Mita, T., Hu, H., 2004, "Nonlinear Tracking Control of A Car-Like-mobile Robot via Dynamic Feedback Linearization". Control 2004, University of Bath, UK.

[16] Zhao, Y., Collins, Jr. E.G., 2005, "Robust Automatic Parallel Parking in Tight Spaces via Fuzzy Logic". Robotics and Autonomous Systems. Elsevier, 51, pp. 111 – 127.

[17] Shladover, S.E.; Desoer, C.A.; Hedrick, J.K.; Tomizuka, M.; Walrand, J.; Zhang, W.-B.; McMahon, D.H.; Peng, H.; Sheikholeslam, S.; McKeown, N., 1991, "Automated vehicle control developments in the PATH program". IEEE Transaction on Vehicle Technology, Vol 40, No 1, Feb.. Pp. 114 –130

[18] Han-Shue, T., Guldner, J., Patwardhan, S., Chen, C. and Bougler, B., (1999) "Development of an Automated Steering Vehicle Based on Roadway Magnets—A Case Study of Mechatronic System Design". IEEE/ASME Transactions on Mechatronics, VOL. 4, NO. 3, September. pp. 258 – 272.

[19] Wada, M., Yoon, K. S, and Hashimoto, H., 2003, "Development of Advanced Parking Assistance System" IEEE Trasactions on Industrial Electronics, Vol. 50, No. 1, February. pp 4 – 17.

[20] Moreno, Edward D.; Penteado, C.G.; Rodrigues da Silva, A.C. Microcontrollers and FPGAs – Applications on Automation Area. Edit. NOVATEC www.novatec.com.br. ISBN 85-7522-079-9, p. 370, 2005.

[21] V. Amudha, B.Venkataramani, R. Vinoth kumar and S. Ravishankar. Software/Hardware Co-Design of HMM Based Isolated Digit Recognition System. In JCP – Journal of Computers, Vol. 4, No. 2, Feb. 2009, Academy Publishers, 2009.

[22] Murakami, Masayuki. Task-based Dynamic Fault Tolerance for Humanoid Robot Applications and Its Hardware Implementation. In JCP – Journal of Computers, Vol. 3, No. 8, Aug.. 2008, Academy Publishers, 2008.

[23] Han, Meng-Ju; Hsu, Jing-Huai; Song, Kai-Tai; Chang, Fuh-Yu. A New Information Fusion Method for Bimodal Robotic Emotion Recognition. In JCP – Journal of Computers, Vol. 3, No. 7, July 2008, Academy Publishers, 2008.

[24] Shimazaki Kazunori; Kimura Tomio; Yamada Satoshi (2004).: Parking assisting device – Patent No US 6711473 B2 - United States.

[25] Tanaka Yuu; Iwata Yoshifumi; Satonaka Hisashi; Endo Tomohiko; Kubota Yuichi; Matsui Akira; Iwakiri Hideyuki; Sugiyama Toru; Kawakami Seiji; Iwazaki Katsuhiko; Kataoka Hiroaki (2006) .: Vehicle backward movement assist device and vehicle parking assist device - Patent No 7039504 – United States.