ABCM Symposium Series in Mechatronics - Vol. 3 - pp.1-10

A CONTROL DESIGN APPROACH USING A RECONFIGURABLE SYSTEM FOR AUTONOMOUS VEHICLE

Anderson P. Correia [Correia, A.P.], anderson_pc@unb.br
University of Brasilia, Campus University Darcy Ribeiro, Dep. ENM, Brasilia/ DF, Brazil

Carlos H. Llanos [Llanos, Carlos H.], llanos@unb.br
University of Brasilia, Campus University Darcy Ribeiro, Dep. ENM, Brasilia/ DF, Brazil

Rodrigo W. de Carvalho [Carvalho, R. W], Rodrigo.unb@gmail.com
University of Brasilia, Campus University Darcy Ribeiro, Dep. ENM, Brasilia/ DF, Brazil

Carla Koike [KOIKE, C. M. C. C.], ckoike@unb.br
University of Brasilia, Campus University Darcy Ribeiro, ICE – CIC, Brasilia/ DF, Brazil

Sadek A. Alfaro [ABSI, Alfaro S. C.], sadek@unb.br
University of Brasilia, Campus University Darcy Ribeiro, Dep. ENM, Brasilia/ DF, Brazil

Abstract. *This paper describes the implementation of a platform based on reconfigurable architecture and concepts of virtual instrumentation applied to the study of 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 FPGA (**F**ield **P**rogrammable **G**ate **A**rray). 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 system 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.*

Keywords*: Reconfigurable Computing, Embedded Processors, Virtual Instrumentation*

1. 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 the 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 simulation/verification tasks during the overall design. Several hardware manufacturers offer design tools, allowing verification/simulation of the electronics system in 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 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 time to reconfigure than *coarse grain* reconfigurable platforms (rDPAs: ***r****econfigurable* ***D****ata* ***P****ath* ***A****rrays* or arrays of rDPUs - ***r****econfigurable* ***D****ata* ***P****ath* ***U****nits*) (Becker and Hartenstein, 2003). 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 partial reconfiguration (PR), 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 languages. The term System on Chip (SoC) (Donecker S. M. *et al.*, 2003) (Yang E. *et al.*, 2004) has been widely used in automation/control system design, communications systems, among others, which involves the use or implementation of different Intellectual Properties (IPs), and a full integration on the FPGA component. 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 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.

In order to test our reconfigurable system technique (used to implement the car controller) and virtual instrumentation based design methodology, the *hands-free driving problem* was chosen. Many researches on vehicle control design as well as **C**ar-**L**ike **M**obile **R**obot (CLMR) (Li *et al.*, 2005) have been done, which apply several techniques based on complex mathematical models (Paromtchik *et al.*, 1998), neural networks (Petko, 2001) (Tzuu *et al.*, 2003), 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 (Giove *et al.*, 2004). 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 **A**utomated **H**ighway **S**ystem (AHS) (Tan *et al.*, 1999).

Some researches have reported the use of FPGA and LabVIEW applied to the either CLMR or hands-free driving problems. A FPGA implementation of a **F**uzzy **G**arage **P**arking **C**ontrol (FGPC) is discussed in (Tan, H.S. *et al.*, 1999). The use of FPGA and LabVIEW is discussed in (National Instruments, 2007) for an accelerator control system design. In (Correia A. *et al.*, 2007a) and (Correia A. *et al.*, 2007b), 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 overall architecture of the system proposed here is described. Section 3 presents basic concepts of the proposed embedded architectural system in the FPGA. Section 4 discusses the defined command set for the control

system as section 5 describes the virtual environment for simulating the vehicle motion. Before concluding, sections 6 and 7 describe our results and conclusions.

2. THE ARCHITECTURE PROPOSED

The overall control system is composed of an embedded control system based on the soft-embedded-processor Microblaze (Xilinx, 2007), which is implemented in a Spartan 3–based FPGA, and a virtual simulator environment implemented in LabVIEW. The architecture is shown in Fig.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.

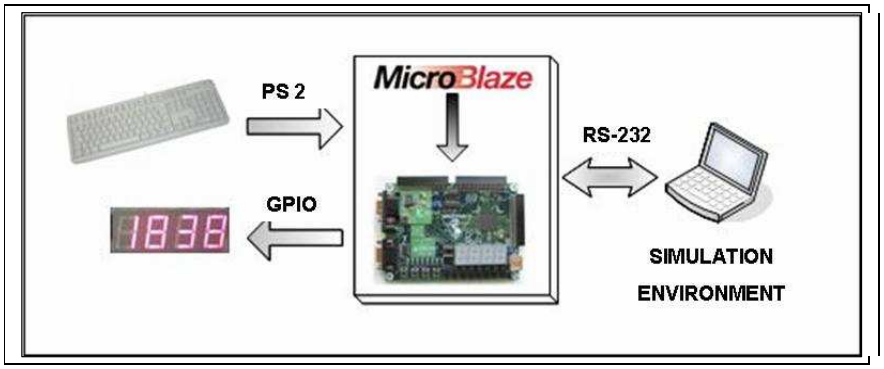

Figure1. The overall System

The embedded microprocessor implements the main control of car tasks in software functions, namely: *break*, *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 options (EDK, 2007), and so on. Finally, a specific keyboard module described in VHDL was added to the design (Correia A. *et al*., 2007a)

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 (Correia A. *et al*., 2007b).

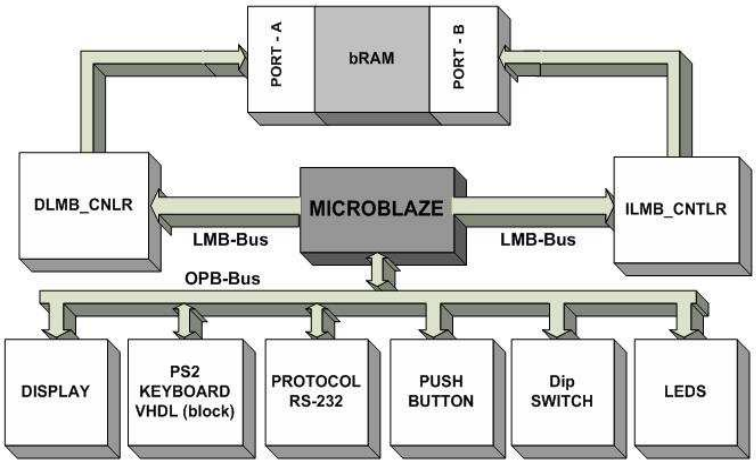

Figure 2. The Hardware System

3. THE FPGA EMBEDDED SYSTEM

The use of FPGAs to implement different type of algorithms is very attractive because these devices offer a trade-off between ASICs (**A**pplication **S**pecific **I**ntegrated **C**ircuits) and general-purpose processors. The control module was defined using the EDK tool (EDK, 2007), in which the Microblaze processor is the system core. This processor has a RISC architecture with 32-bit general purpose registers, an **A**rithmetic **L**ogic **U**nit (ALU), 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 shows the architecture of the control system, 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 Fig. 2).

3.1 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:

a) **The *break.c* module:** it receives a defined command to operate the car-break (see section 4). The module verifies what is the current position of the brake and it gives the proper direction to the actuator. A PWM (**P**ulse-**W**idth **M**odulation) 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 4) 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 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.

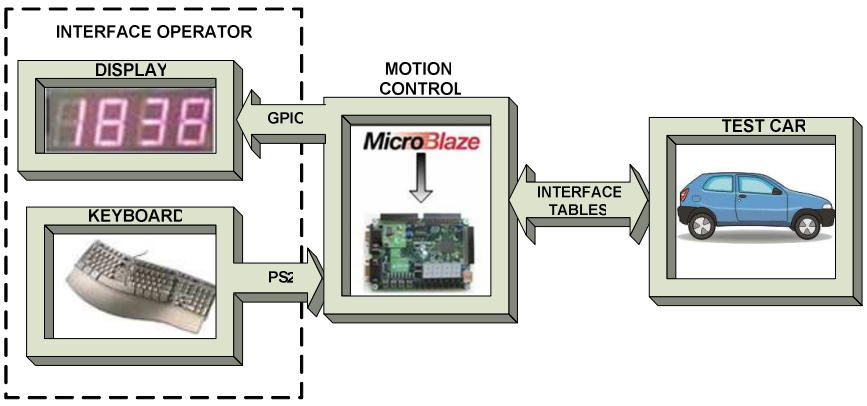

Figure 3. Overall Hardware System

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* axes in a predefined way. The DC-motor's speed is controlled by two PWM-signals.

3.2 The Hardware Modules of the Controller

The hardware modules are depicted in Fig. 3. The led, display and pushbutton modules were automatically generated by the EDK system. On the other hand, the keyboard module was first described in a VHDL 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 (Correia A. *et al*., 2007a) and (Correia A. *et al*., 2007b).

4. THE COMANDS 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 1 and table 2. The commands are organized into two sets, describing both *manual* and *automatic* modes (see tables 1 and 2, 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 2). For example, the command related to the clutch (EB*xy*) 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 2). The parameter *y* is only valid for the third case (EB*Cy*, see table 2), 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.

Table 1: 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

Table 2: Commnand-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*) and 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.

5. THE SIMULATOR ENVIRONMENT

To model the vehicle kinematics, the three canonical equations describing the positioning in the Cartesian *x and y* coordinates of the nonholonomic vehicle were applied (Tan H.S. *et al.*, 1999). The program was designed by means of several software modules, involving the RS232 interface, the car design, the new car position calculation and the user interface. Some parts of the calculation module were directly implemented in C language in order to represent the equations. Figure 4 shows a part of throttle control of the virtual car. Additionally, a module for the kinematics equation implementation (written in C language into the LabVIEW program) was implemented, which allows to represent the vehicle movement in real time. Other modules for achieving RS-232 communication, clutch, break and gear behavior were also included in the system.

The user's interface of the simulator environment is shown in Fig. 5 and it represents the car position and several blocks for monitoring the current engine rotation, gear position, throttle, among others. The main task of the vehicle simulation module is the simulation of 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. This module is responsible for manipulating the virtual car model.

The modules are composed of the corresponding control parts: steering wheel, brake, gear, clutch and throttle. Furthermore, the simulator environment sends to the controller (via RS-232 based protocol) the current status of the control variables such as potentiometer-position for monitoring the status of the break, steering wheel, gear and throttle. These variables are represented using 8-bit words.

6. RESULTS

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.

6.1 The FPGA Synthesis Results

The FPGA synthesis results were obtained in the EDK project report (Xilinx, 2007). 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.

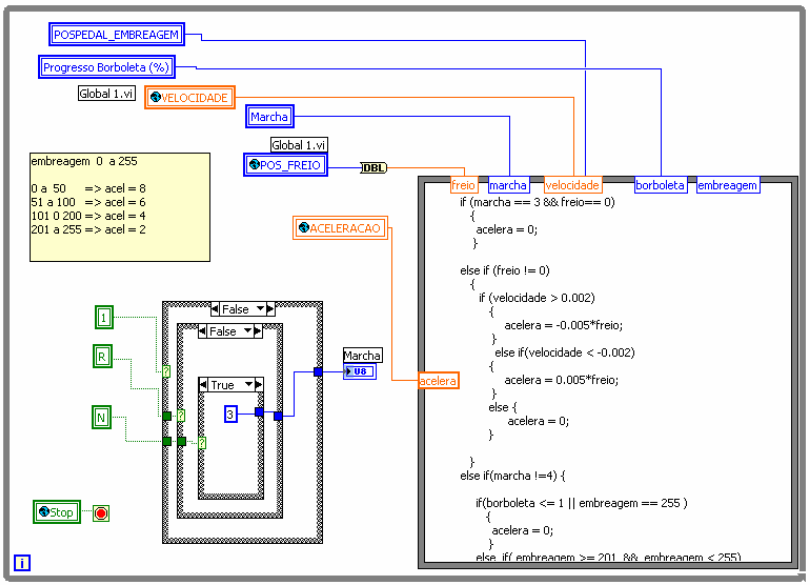

Figure 4. Software Structure of the Virtual Simulator Environment – the Throttle Control

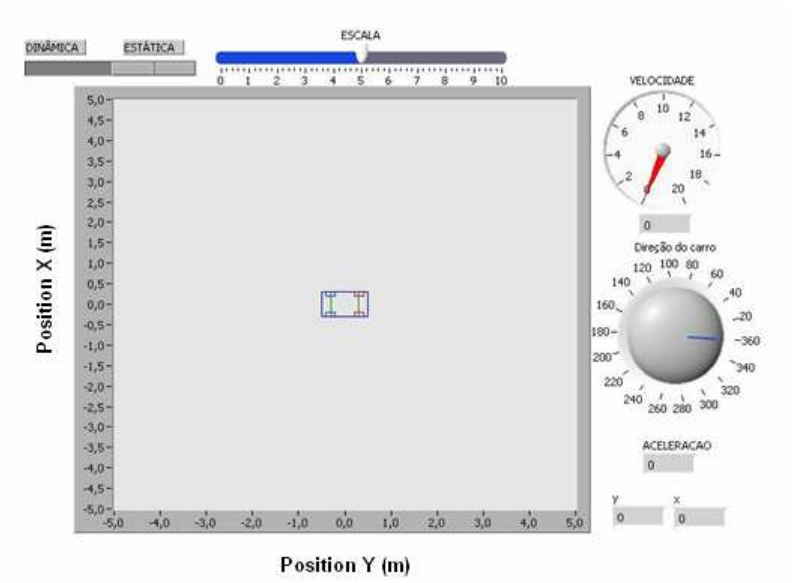

Figure 5. The user interface in the LabVIEW

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). 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 3 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 3: Synthesis results in the EDK tool

Module	Slices (%)	Slices flip-flops (%)	LUTs (%)	Bounded IOB	BRAM (%)	Max Frequency (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_7 segled_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

6.2 The Simulation Environment Results

The simulator environment results are shown in Fig. 6 as a sequence of illustrations demonstrating the vehicle movement controlled by the same commands shown in tables 1 and 2.

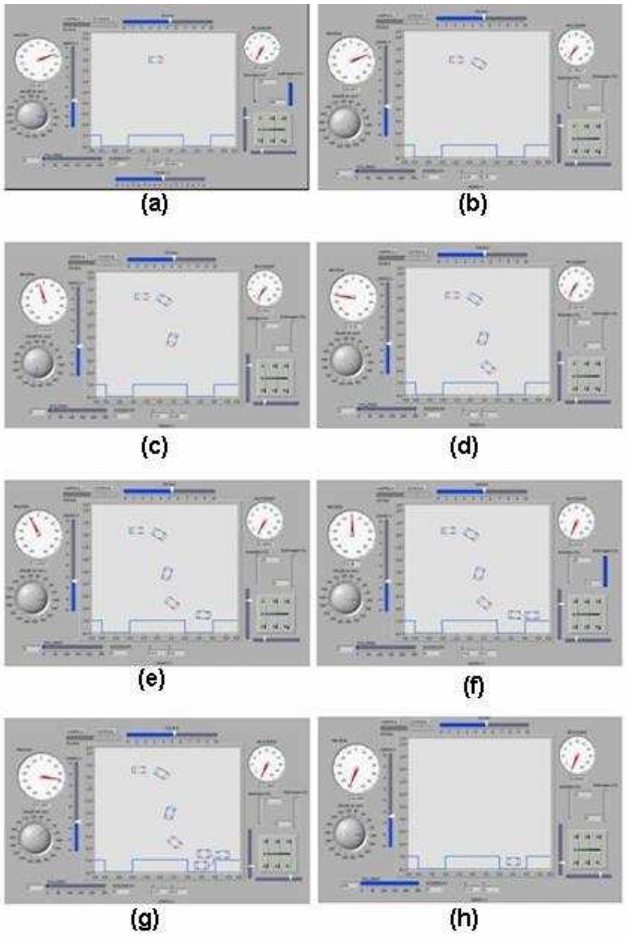

Figure 6. The Simulator Environment Results

Figures 6.a to 6.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 6.h shows the final position of the car. All the commands were of the manual-mode set (see table 1).

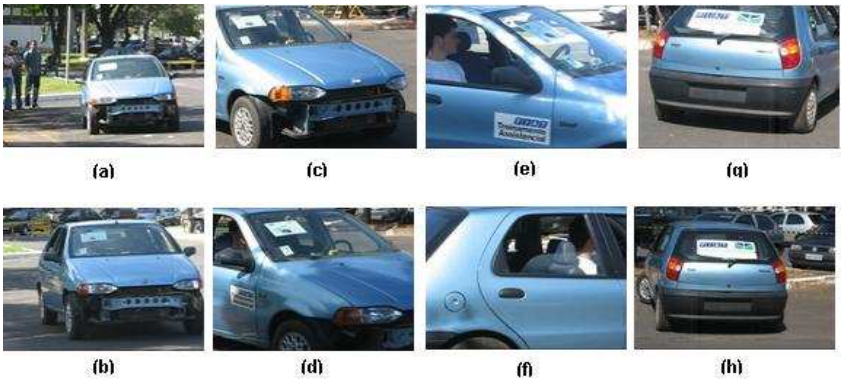

Figure 7. Results of Car Test

6.3 The Testing Car Results

Tests of car movement control were accomplished with the objective of validating the movement controller. The vehicle was controlled through commands sent through the keyboard. Figure 7 shows a sequence of images of the film of the car movement test. In Figure 7.a the car is stopped whereas Fig. 7.b shows the car beginning the movement (the processes to accomplish the first-gear was already finished). Several maneuvers are shown in Fig. 7.c to 7.h, which imply the use of the clutch, the steering wheel and the break. All the commands (see Tab. 1 and Tab. 2) were sent to the FPGA embedded controller through the keyboard.

6.4 Simulator-results/Real-car-result (comparing the results)

Figure 8.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, Fig. 8.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, see Fig 8.a and Fig. 8.b).

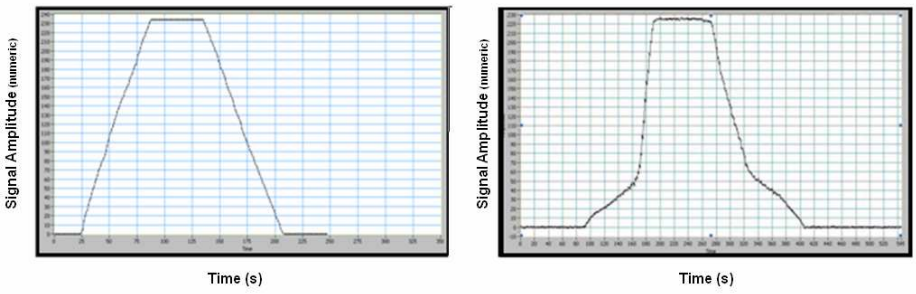

Figure 8. (a) The throttle signal in the simulator. (b)The throttle signal in the car

7. CONCLUSIONS

A flexible environment for validating/simulating complex mechatronic systems was developed using the virtual instrumentation approach. This approach was applied for studying the hands-free driving automobile problem, including a controller based on a reconfigurable architecture. The car control was implemented using the Microblaze embedded processor. A serial-based communication protocol was defined in order to control the car motion, which includes the steering wheel, clutch, gear, break and throttle subsystems. Additionally, a protocol was defined and tested for allowing the user to send commands to the controller (typed in a keyboard), and the controller sends predefined data packages to the LabVIEW environment in order to update the current status of the car in real time.

The tests in the simulator and in the 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 suited devices for implementing several automation and control techniques due to the fact that allow for embedding both typical microprocessor such as ARM family (Altera, 2007) 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 validating and simulating solutions for several problems in the robotic and mechatronic areas (Dudek G. and Jenkin M., 2000).

8. REFERENCES

Altera, 2007. Available at http://www.altera.com. Accessed in 2007.

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

Correia A.; Llanos C. H. Q.; Carvalho R. W.; Alfaro S. A. (a), 2007, "A Design/Testing Platform Based on Reconfigurable Architectures and Virtual Instrumentation Applied to the Hands-free Driving Automobile Problem". WSEAS Transactions on Systems and Control. Issue 3, vol. 2, pp. 297 – 304.

Correia A.; Llanos C. H. Q.; Carvalho R. W.; Alfaro S. A. (b), 2007, "A Platform Based on Reconfigurable Architectures and Virtual Instrumentation Applied to the Driving Automobile Problem". 6th WSEAS International Conference on Signal Processing, Robotics and Automation (ISPRA '07). Greece, February 2007. pp. 1– 8.

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

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

EDK, 2007, "Platform Studio, User Guide". Available at http://www.xilinx.com/ise/embedded/edk_docs.htm. Accessed in 2007.

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

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.

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.

National Instruments, 2007. Available at http://www.ni.com./labview/whatis/. Accessed in 2007.

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 2007.

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

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.

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

Xilinx. Inc, 2007. Available at http://www.xilinx.com/. Accessed in 2007.

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.

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.

Acknowledgments. This work is partially supported by FINATEC (Fundação de Empreendimentos Científicos e Tecnológicos), UnB and CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), Brasília/DF, Brasil.