A Platform Based on Reconfigurable Architectures and Virtual Instrumentation Applied to the Driving Automobile Problem.

Anderson Correia[1], Carlos H. Llanos[1], Rodrigo W. Carvalho[1] and Sadek A. Alfaro[1]
[1] Department of Mechanical Enginneering – GRACO
Univerity of Brasília
Campus Universitário Darcy Ribeiro, Faculdade de Tecnologia, Brasília, Distrito Federal
Brazil

Abstract: - This paper describes the implementation of a control system for hands-free driving car based on reconfigurable architectures and the virtual instrumentation concepts. The reconfigurable architecture approach means the use of an embedded microprocessor (Microblaze, Xilinx) jointly with several hardware modules, which were described in VHDL hardware description language. The virtual instrumentation approach refers to the use of LabVIEW environment in order to develop a simulation/validation tool, suitable for several mechatronic applications. This work is focused on a vehicle control system design and its validation using a LabVIEW program. The control system was developed for solving a well-known problem: the hands-free driven car. The car control (implemented in the FPGA) and the LabVIEW program are communicated by means of a RS232 interface. A protocol was defined allowing the user to send defined commands to the controller (typing commands in a keyboard). This approach opens a wide variety of possibilities for validating and simulating solutions for several problems in the robotic and mechatronic areas.

Key-Words: - Embedded Processors, Hands-free driving vehicle, Virtual Instrumentation.

1 Introduction

This paper describes the implementation of a control system for hands-free driven car based on reconfigurable architectures and the virtual instrumentation concepts. The reconfigurable architecture approach means the use of an embedded microprocessor (Microblaze, Xilinx) jointly with several hardware modules, which were described in the VHDL hardware description language. The virtual instrumentation approach refers to the use of LabVIEW environment in order to develop a simulation/validation tool, suitable for several mechatronic applications. This work is focused on a vehicle control system design and its validation using a LabVIEW program. The control system is developed for solving a well-known problem: *the hands-free vehicle driving*. Many researches in hands-free vehicle problem as well as *Car-Like Mobile Robot* (CLMR) [3][10] have been done, which apply several techniques based on complex mathematical models [11], neural networks [5][6], genetic algorithms, fuzzy logic [13], among others. Steering a car is confined with conditions of 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 [1]. 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 system is called as an *Automated Highway System* (AHS) [9].

Given the complexity of the hands-free driving problem It is very important to define a design environment that just allows the testing and validation of different control strategies, apart from a rapid prototyping of the electronic control system. In the last years embedded systems has been investigated for automotive industry applications, especially by using reconfigurable architecture approaches. Reconfigurable architectures are based on the use of processors (based on the von Neumann Model), which are implemented in FPGAs (**F**ield **P**rogrammable **G**ates **A**rrays), jointly with several hardware parts described through HDLs (**H**igh-Level **D**escription **L**anguages).

FPGAs devices provide high performance for parallel computation and enhanced flexibility (if compared with ASICs implementations) and are the best candidates for several kinds of hardware implementations. The FPGA can be configured by means of software tools, allowing the easy implementation of complex systems such as those related to control/automation applications, communications, parallel computing, among others.

Virtual instrumentation combines mainstream commercial technologies [7], such as the PC, 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. Virtual instrumentation approach has been widely used in the context of prototyping of automation/control systems. National Instruments LabVIEW software uses symbolic/graphical representations to speed up the system development of instrumentation systems. The software symbolically represents functions through icons. Additionally, the environment permits the representation in real time of system's processes, allowing the designer the rapid validation of the results.

The objective of this work is to study and solving the *hands-free driving* automobile problem using reconfigurable architectures and virtual instrumentation. Our strategy distinguishes oneself from the classical design flow because of the FPGA-based control system design and the use of LabVIEW environment, in order to represent the current status of the vehicle even in real time. The car control was implemented using the Microblaze embedded processor [12]. The control and the LabVIEW program are communicated by a RS232 interface, in which was developed a communication protocol. The protocol was defined for allowing the user to send commands to the controller (typing 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.

A few researches have been reported in the use of FPGA and LabVIEW applied to the CLMR or hands-*free driving* problems. A FPGA implementation of a *Fuzzy Garage Parking Control* (FGPC) is discussed in [10]. Otherwise, the use of FPGA and LabVIEW is discussed in [4] for an accelerator control system design.

In section 2 the overall architecture of the system is described. Section 3 presents the basic concepts of the proposed embedded architectural system in the FPGA. Section 4 discuses the defined command set for the control system. Section 5 describes the virtual environment for simulating the vehicle motion. Section 6 describes the communication protocol. Before concluding, section 7 describes our results.

2 The proposed Architecture

The overall control system is composed of an embedded control system based on the soft-embedded-processor Microblaze [12], 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 (that is implemented in the FPGA).

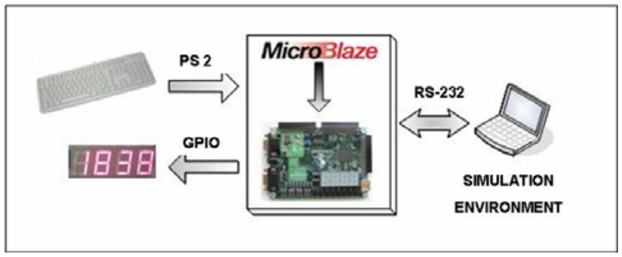

Figure 1: The overall System

The embedded microprocessor implements the main control car tasks in software functions, namely: *break*, *clutch*, *steering wheel*, *gear* and *throttle* sub-systems of a real vehicle. Each function was described in *C* language in a structured software approach. Several hardware modules were incorporated to the hardware design such as RS232, buttons, display using the EDK tool options [2] during the project specification. Finally, a specific keyboard module that was described in VHDL was added to the design.

2.1 The Keyboard Interface

The user can send commands to the controller (implemented in the Microblaze embedded microprocessor). The keyboard can be substituted easily by a joystick (see section 4).

2.2 The FPGA Embedded Controller

The controller is implemented in the Microblaze embedded microprocessor, which run several software functions implemented in C language (see section 5).

2.3 The Simulator System (LabVIEW)

The 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 (see section 6).

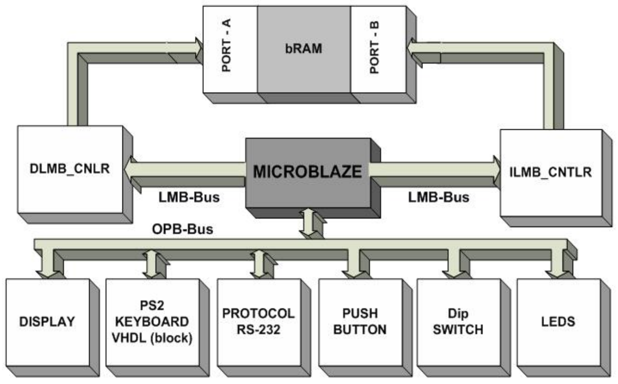

Figure 2: The Hardware System

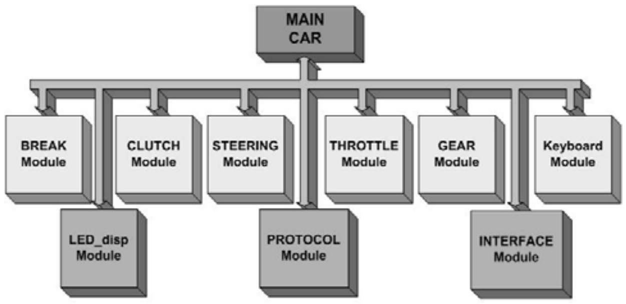

Figure 3: Software Project of the Controller

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 [2], in which the Microblaze processor is the system core. This processor has a RISC architecture with thirty-two 32-bit general purpose registers, an **A**rithmetic **L**ogic **U**nit (ALU), a shift unit and interrupts, among others possible peripherals.

The EDK tool is a embedded development environment that includes a library of peripheral IP cores, where the *Xilinx Platform Studio* tool suites for intuitive hardware system creation. Additionally, a Built-On Eclipse software development environment, GNU compiler and a debugger are included as well. Figure 2 shows the architecture of the control system, which was synthesized using the EDK. The communication of the processor with peripherals 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 2).

3.1 The Software Modules of the Controller

Once the processor system was configured and your peripheral were defined all the programming was made in standard C, compiled and tested inside of the EDK environment.

The software modules were described in a structured way, whose block diagram is depicted in figure 3. 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 is 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 what is the current position is and it gives the proper direction to the wheel actuator.

d) **The *throttle.c* module**: It works in too ways: the first one works for controlling the butterfly position, which is represented 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.

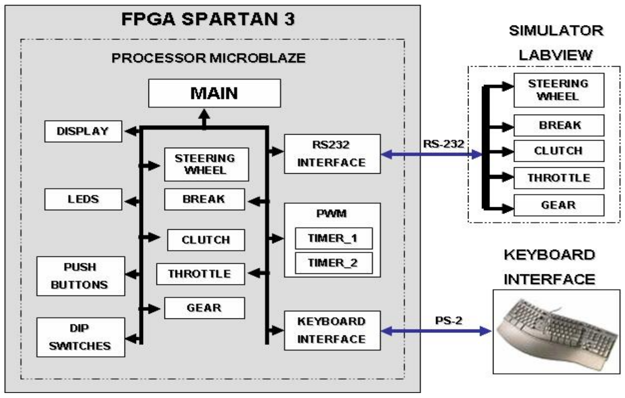

Figure 4: Overall Hardware System

e) **The *gear.c* module**: this module receives the command of the operator (see section 4) and verifies the current position for exchanging the gear-position. This is achieved by two DC-motors, which moves 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 figure 4. The led, display and pushbutton modules were automatically generated by the EDK system. On the other hands, the keyboard module was described in a VHDL file, which implements the PS2 protocol and afterwards it was incorporated 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 depending on the necessities of the system. In this case, only two PWM modules have been generated. Other PWM modules can be easily added to the design using the EDK environment. Each timer has two programming registers, namely TCSR0 and TCSR1, which are used for implementing different functions, depending on the programming modes (generate, capture and PWM modes).

4 The Commands for the Control System

Several commands were defined in order to control the car and its definitions with a specific syntax and semantic are described in table 1. The commands are organized into two sets, describing *manual* and *automatic* modes. 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. Other manual commands can be seen in table 1.

The second mode presents commands for using either via keyboard or into the C program. Each command was implemented in a specific C function. For example, the commands related to the clutch (*EBA*, *EBR* and *EBC*) have specific syntax/semantic. In this case, *EBA* and *EBR* commands have not parameters as long as *EBC* needs one parameter, which represents a value among 0 to 100% of the total clutch position.

The commands are put by the user using the keyboard and then the Microblaze identifies and processes them, before to send the appropriate control signals (to the actuators) using the RS232-base protocol (see section 6).

Table 1: command definition for the car control

Activated F1 (Manual-Mode)		No Activated F1 (Automatic-Mode)	
Command	Action	Command	Action
↑	Increases the rotation of the motor	DID val	Steering right. val describes the angle (0 to 60)
→	Rotates the wheels for the right	DIE val	Steering left. val describes the angle (0 to 60)
←	Rotates the wheels for the left	DIC	Steering to center
↓	Reduces the rotation of the motor	CAB val	Gear (0 to 7): 0 means neutral position, and 7 means reverse)
Space	Brake	EBA	Completed push enable of the clutch
N	Neutral	EBR	Fast clutch disable
R	Reverse gear	EBC val	Controlled Clutch disable (by step-motor, val 0 to 100%)
1	First gear	FRE val	Break (val 0 or). Set or release the break
W	Increasing the clutch	ACB val	Throttle (val: 0 to 100).It Defines the Butterfly position
S	Reducing the clutch	ACR val	val : define the motor rotation

5 The Simulator System Environment

LabVIEW is a general-purpose graphical programming system with extensive libraries of functions for any programming tasks. In addition, this system includes libraries for data acquisition, instrument control, data analysis, data presentation, and data storage.

For the modeling of the vehicle kinematics there were applied the three canonical equations that describe the positioning in *x and y* of the nonholonomic vehicle [10]. The equations (1, 2 and 3) define the path and the position in *x* and *y*. The equation 3 describes the instantaneous position of the angle of the wheel.

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

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

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

The program was designed by means of several software modules, involving the RS232 interface, the car design, the new position calculation and the user interface. Some parts of the calculation module were directly implemented in C language in order to implement the equations. Figure 5 shows a part of the block of RS232-based serial communication. Additionally, there were implemented modules for the kinematics equation implementation (which were implemented in C into the LabVIEW program) and the real time vehicle movement implementation.

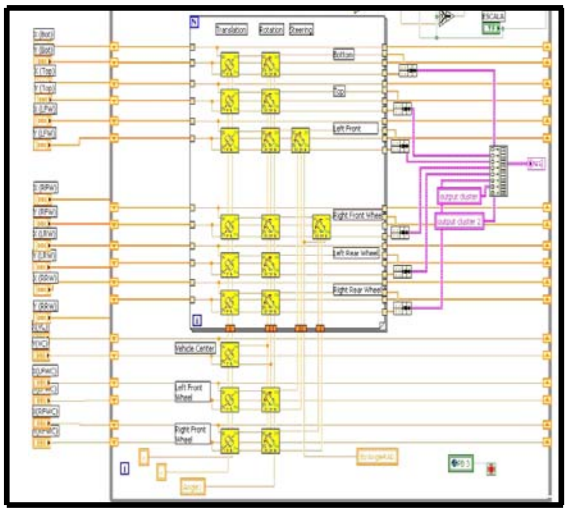

Figure 5: Software Structure of the Virtual Simulator Environment – Kinematics Control

User's interface is shown in figure 6 that 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. It was applied the concepts of *Virtual Instrumentation* by programming in LabVIEW environment in order to generate the appropriated signals, depending on the control and status variables. This module is responsible for manipulating the virtual car model, which is composed of the corresponding control parts, namely steering wheel, brake, gear, clutch and throttle.

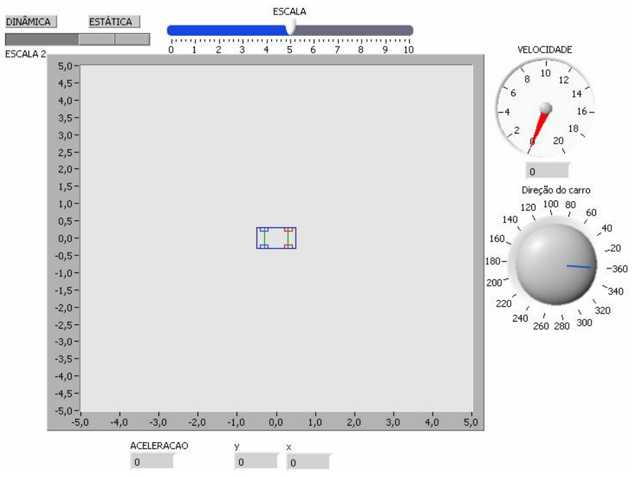

Figure: 6 The User interface in the LabVIEW

6 The communication protocol

A communication protocol was defined in order to implement a full-duplex communication between the control module (FPGA) and the LabVIEW program (see figure 1). The controller sends to the LabVIEW a 3-bytes package, where the first one represents a specific car *state-variable*, encoded in 3 bits (namely, *front-wheel_1*, *front-wheel_2*, *x-position of the gear*, *y-position of the gear*, *break-position* and *clutch-position*). The second byte represents the information for controlling the clutch, in which the 4-most-significant bits are used for generating 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 throttle (2-bits), steering wheel (2-bits) and break (2-bits).

For example, the first bit of the throttle is used to represent the direction and the second is used for generating the proper PWM signal. Given that the packages are sent in a sequence, the LabVIEW is capable to rebuild the PWM signal using only one bit in the serial communication.

In the case of the LabVIEW, It sends to the controller the state-information related with several variables. To achieve this, a 2-bytes package is used. 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, *break-position* and *clutch-position*), by using the 3-most significant bits.

The second byte represents the current value of the state variable. In this case, the LabVIEW program responds to the controller about the required state information.

7 Results

The FPGA synthesis results were obtained in the EDK project report. The results are shown in table 2 for the main modules of the control system. It was used a Spartan 3 device (xc3s200ft256-4) for the hardware implementation of the controller. The main resources consumption is related to the Microblaze implementation. The clock frequency is depicted 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, and the critical frequency is for the 7-segment driver (about 68 MHz). The same table depicts only a peripheral implementation for PWM signal. However, other PWM devices can be

Table 2: Synthesis results in the EDK tool

Module	Slices (%)	Slices flip-flops (%)	4-input LUTs (%)	Bounded IOB (%)	BRAMs (%)	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_3Bit	2	1	2	64	0	138.927
interface_io	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_io	2	1	0	98	0	138.658
pwm_timer_0	13	8	7	67	0	98.348

easily added in the design depending on the design's requirements. The software implementation of the controller (written in C and compiled) is stored in the *Bram-bloc* (second line of the table 2). In this case the total FPGA-RAM-elements was 66%.

The LabVIEW program was capable to represent the kinematics and behavior of the vehicle in real time and a realistic way. This performance includes the serial communication (with the implemented protocol), the car position calculation and the computational cost of the system interface in the PC. The baud-rate of serial communication can be changed in both programs: controller (in the Microblaze) and the LabVIEW. The current baud-rate of the serial communication is 9600 bauds. The PWM-signals are transmitted through the RS-232-based protocol and the movement of the vehicle is shown in the control-panel of the LabVIEW program taking into account this information.

8 Conclusion

A flexible environment for studying the hands-free driving automobile problem was implemented based on reconfigurable architecture and virtual instrumentation tool (LabVIEW). 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 subsystem. Additionally, the protocol was defined and tested for allowing the user to send commands to the controller (typing 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.

Otherwise, this approach opens a wide variety of possibilities for validating and simulating solutions for several problems in the robotic and mechatronic areas [8].

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.

References:

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

[2] EDK: Platform Studio, User Guide. Available at http://www.xilinx.com/ise/embedded/edk_docs.htm. Accessed in November of 2006.

[3] Baltes, J., and Lin, Y. Lin: Path- Tracking Control of a Non-Holonomic Car-like Robot with Reinforcement Learning. CITR, Tamaki Campus, University of Auckland, (1999) 1–17

[4]Giove, D., Martinis C. D., Mauri, M.: Reconfigurable Hardware Resource in Accelerator Control System. EPAC, Lucerne, Switzerland (2004) 701 – 703

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

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

[7] National Instruments. Avail. http://www.ni.com./labview/whatis/ Acc. in 2006

[8] Petko, M., Uhl, T.: Embedded controller design-mechatronic approach. IEEE, Second Workshop on Robot Motion and Control. (2001) 195-200

[9] Tan, H.S., Guldner, J., Patwardhan, S., Chen, C., Bougler, B.: 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 (1999) 258 - 271

[10] Tzuu-Hseng, S., Chang, S-J., Chen, Y-X.: 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 (2003) 3776 – 3781

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

[12] Xilinx. Inc. Available at http://www.xilinx.com/ Accessed in 2006

[13] Zhao, Y., Collins, Jr. E.G..: Robust Automatic Parallel Parking in Tight Spaces via Fuzzy Logic. Robotics and Autonomous Systems. (2005) 111 – 127