A Framework for Implementing Intelligence in Embedded Controls

S. Commuri, *Member, IEEE*

Abstract—This paper presents a framework for the realization of intelligent embedded systems for control applications. Requirements on system intelligence are translated into requirements on the system hardware and software. The hardware and software components of the design and their impact on system performance are discussed. Software and hardware architectures are presented that enable the realization of these requirements. Recent trends in the hardware-software co-design and hardware reconfigur-ation are utilized to design systems that are fault-tolerant and have higher reliability under all operating conditions. The design approach is validated by means of a case study.

I. Introduction

In recent years, performance specification for control systems has grown to include the requirements for system intelligence. These systems are typically distributed in nature, require a high degree of fault tolerance, and have to function under varying operating conditions. These systems also have to support intelligent sensor selection and sensor fusion, remote monitoring and operation, and be capable of implementing sophisticated algorithms for adaptive behaviors. Ultimately, the design should support system evolution, provide paths for migration of capabilities and result in lower acquisition and lifecycle costs. Design of such systems cannot be accomplished by the simple integration of smart controllers or transducers [1,2]. Embedding intelligence into systems requires a new design paradigm that takes into account the hardware and software complexities involved in the design of embedded systems [3],[4]. Thus, it is necessary to design the systems ground-up in order to meet the system requirements [5], [6].

Researchers have addressed the problem of controlling intelligent systems from many perspectives. Efficient use of resources was accomplished using a centralized planning approach [7], [8]. An approach based on distributed controls was adopted to meet the requirements of fault tolerance and fault recovery [9], [10].

This work is supported in part by the U.S. Department of Defence, Army Research Office, under grant # DAAD 19-03-1-0142.

S. Commuri is with the School of Electrical and Computer Engineering, University of Oklahoma, Norman, OK 73019, USA (e-mail: scommuri@ou.edu)

As real time requirements for such embedded systems grew more stringent, researchers considered both centralized and distributed architectures in their design [11]. Distributed systems make possible the development of sophisticated systems with complex behaviors. In such systems, time-critical reactive behaviors of the system can be implemented locally, while generalized system-level behaviors can be abstracted out and implemented on a central resource that communicates with all the distributed nodes in the system. Such implementations encourage modularity in the design and facilitate fault-tolerant design. Crucial in the design of such systems is the selection of the appropriate hardware and software components and the architecture for their integration.

While the component technologies have matured in the past few years, the design of these systems is not a matter of simple system integration. Often, it is necessary to design these systems ground-up in order to meet the overall requirements. In the conventional design process, the functionality expected from the overall system is specified during the specifications stage of the project. These specifications then drive the design of the system. The performance targets specified are used to allocate hardware resources and determine the software requirements. Once the hardware allocation is made, system performance can be altered only by modification of the software. Thus, plug-and-play capability of sensors/actuators, fault tolerance, changing the system functionality, etc. can be achieved only by planning redundancies in the hardware and reconfiguring the software during run-time to meet the changing requirements. Therefore, the key design issue to be addressed in this approach is the compromise between the system performance and its cost and flexibility [5], [6].

An alternate approach is to use reconfigurable hardware components, like Field Programmable Gate Arrays (FPGAs), to allow for both hardware and software reconfiguration [12] - [15]. By systematically partitioning the system, functionality requiring changing execution paths or those that impact performance can be assigned resources that allow both hardware and software reconfiguration. Successful implementation of this technique requires the development of new system architectures for Reconfigurable Computing. Reconfigurable Computing is the ability of the software to reach through to the hardware and alter the data path for execution thereby optimizing the performance. In this paper, a hierarchical architecture that allows for plug-

0-7803-9419-4/05/$20.00 ©2005 IEEE

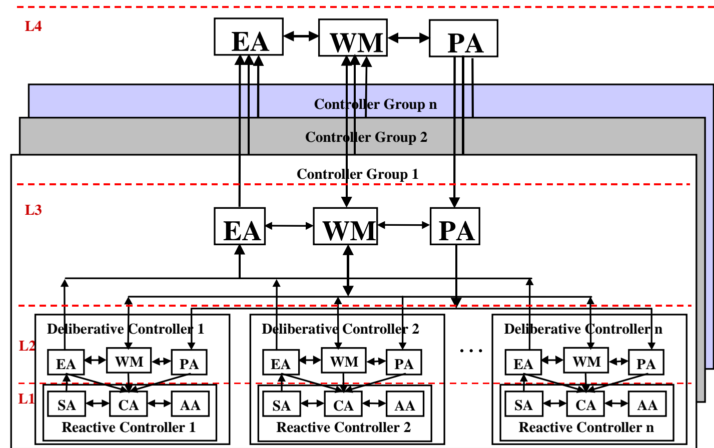

Fig. 1. Architecture for the implementation of embedded controllers for intelligent control applications.

and-play and fault tolerance at the lowest level and for learning and adaptive behaviors at the highest level is proposed. The features of this architecture are then exploited in the design of reconfigurable hardware and software modules that aid in the implementation of embedded controllers for intelligent systems.

The rest of the paper is organized as follows: Section 2 presents the architecture for the implementation of an embedded controller for an intelligent system. The requirements on system intelligence are then translated into requirements on the system hardware and software. In Section 3, the design methodology is discussed in light of the recent trends in the hardware-software co-design and hardware reconfiguration that enable the realization of these requirements. Examples are provided in Section 4 to illustrate the design approach.

II. SYSTEM REQUIREMENTS AND ARCHITECTURE

Recent advances in computers and computer networks have resulted in the proliferation of low cost, distributed systems. The impact of these advances on the embedded controllers has been significant and their performance specifications now include the need for remote operation, coordination with other controllers, and built-in provision for system upgrades. These systems must also support intelligent sensor selection, sensor fusion, increased fault tolerance, and adaptive behaviors. Designing such systems is substantially more complex than the simple integration of smart sensors and actuators. In fact, such integration may impose severe restrictions in the communication between the different modules in the system and will likely lead to degradation in the performance. Therefore, embedding intelligence into a system involves developing design paradigms that takes into account hardware and software complexities [3], [4].

Embedded controllers are characterized by stringent real-time requirements. On the other hand, system intelligence is distributed in nature, requires modular components that allow for plug-and-play, modification-on-the-fly, etc., all of which are not amenable to tight real-time behavior. To overcome this contradiction, an architecture is presented that is hierarchical in nature and allows system intelligence to be incorporated at all levels of the hierarchy. The design methodology developed in this paper enables the design of simple components whose performance can be rigorously analyzed. Complex hierarchical systems can them be constructed using these low-level building blocks. The proposed architecture is shown in Fig. 1.

In Fig. 1, the lowest layer, i.e. Layer 1 (L1) in the system consists of the sensing, actuation and control functions. In this layer, an embedded controller will typically have a control agent (CA), an actuator agent (AA), and a sensor agent (SA). The control agent is responsible for attaining the commanded system performance at the lowest level. It can command the sensor agent to override its output values, recalibrate its signal, as well as perform rudimentary signal processing like filtering. The AA and SA have the lowest level of autonomy and are completely controlled by the control agent. This layer is characterized by low intelligence, stringent real-time requirements and fault-tolerant behavior.

Layer 2 (L2) of this framework provides higher level of abstraction within the system. At a very fundamental level, this design is adequate for an embedded controller to function and perform simple control tasks in a structured environment. Note that, because of our distributed communication infrastructure, the sensor, actuator, and control resources (and the corresponding sub-agents) for a single system need not be present on the same physical platform.

In order to meet the requirements of fault tolerance, uncertainty in the system model and the environment, we propose a distributed architecture wherein the higher layer (L2) incorporates elements that instill higher-level intelligence in the system. In this layer, the sensory signals from Layer L1 are processed by the Estimator Agent (EA). The output of the Estimator is then used to modify/update the local representation of the World Model (WM) and as input to the Control Agent. The distributed intelligence paradigm that is proposed means that EA can now include algorithms for fault detection, dynamic sensor reconfiguration, and sensor fusion. The WM entity maintains information about the environment that is necessary for the successful tasking of the system. The Planning Agent (PA) utilizes the information from the local model of the world (WM) and the high-level task requirements to generate a plan that is communicated to the control agent in layer L1. Level L2 is characterized by increased autonomy and less stringent real-time requirements. It is to be noted that the architecture specified is independent of hardware and software implementations and individual elements in L2.

A network of distributed controllers may consist of a number of individual controllers possibly with differing sensor/actuator suites and capabilities. The coordination between these controllers is managed by the PA entity at the level of the controller group (L3). Information sharing between L2 entities is controlled by the entities in L3. This increases the security of the implementation because the L2 entities can function independently of each other, while still functioning in a coordinated manner. The primary function of the entities in Layer 3 is to coordinate the working of the individual controllers in the group. L3 handles all reassignments of tasks between different controllers in L2. Introduction of new controllers or sensor suites, etc., are the exclusive domain of L3. The outputs of all the EAs in layer L2 provide the input to the EA module in L3. Team-level sensor fusion amongst the different controllers is accomplished by the EA at L3. This EA module is used to update the world model (WM) in Layer 3. This WM also manages the information sharing among the different controllers in L2. The planning agent (PA) in this layer does the task decomposition from the overall system requirements and updates the individual PAs in L2. It is to be noted that the architecture specified is independent of hardware and software implementations and individual elements in L2. Layer 4 (L4) manages the coordination between groups of robot agents. The highest level of intelligence and autonomy and the lowest level of real-time criticality characterize L4. Dynamic reassignment of the responsibilities of each group is handled by L4.

The proposed architecture will enable the development of groups of distributed controllers that are "intelligent." The architecture is flexible and is not dependent on the type of controllers or algorithms implemented in any given layer.

III. Hardware and Software Architectures for Intelligent Systems

In this section, we develop the features of the embedded controllers that are suited for use in the framework proposed in Section II. Embedded controllers are typically implemented as specialized algorithms running on a microprocessor based system. While general purpose microprocessors afford flexibility in the design of the software, this advantage is usually offset by inefficiencies in the implementation. For tasks requiring high degrees of efficiency, Application Specific ICs (ASICs) that are specifically designed for the application can be used. However, this performance comes at the price of flexibility and makes it harder to meet the system objectives like reconfigurability and fault tolerance. Software modules allow for changing system functionality but the performance obtained is restricted by the choice of the hardware platform. We propose a solution based on reconfigurable hardware and software modules to meet the system requirements.

A. Hardware for Reconfigurable Computing

Reconfigurable Computing is the ability of the system to reach through the hardware layer and change the data path for execution. Such reconfigurable elements can adapt dynamically to be compatible with an ad-hoc network of associated components or adapt to changes in associated components. In order to harness the power of hardware reconfiguration, the system should be partitioned to tasks executable in software and hardware. Any task requiring high computations and real time execution can be implemented in hardware while tasks for general purpose computing can be implemented in software. Field Programmable Gate Arrays (FPGAs) offer an attractive method for implementing hardware that can be configured to meet the needs of the application. Advances in FPGA fabrication technology and availability of millions of gates in a single chip allows for building hardware solutions that were earlier realized only in software. The types of reconfiguration possible with these devices and the specific system requirement met are now discussed.

Full vs. Partial Reconfiguration:
Built in Self Test is the first task executed on system startup to verify proper functionality. On successful completion of the startup sequence, the system transitions into an operational mode. Traditionally, the type of test sequences that could be run was limited by the functionality of the hardware. This limitation can be addressed by harnessing the power of reconfigurability of a FPGA. Here, a first configuration is loaded for self test and on successful completion a run-time

configuration is loaded onto the FPGA device. This type of reconfiguration is called Full Reconfiguration. Since the system can be optimized for every task separately, the overall performance is increased. Often, a system requires only a portion of its functionality to be changed, especially during fault recovery where there might be a need to reconfigure only the sensor module or simply bypass the sensor. This can be done using Partial Reconfiguration. Partial Reconfiguration is supported by some FPGAs where a portion of the circuitry is reconfigured while the rest of the device is unaffected and still in operation.

Static vs. Dynamic Reconfiguration:

Static Reconfiguration is the process where the system has to be taken offline and configured before it goes into operation. On the other hand, dynamic reconfiguration can take place while the system is under operation. However, care has to be exercised to prevent changing portions of the hardware during execution to prevent unforeseen outcomes. Dynamic reconfiguration is essential when it is not feasible to take the system off-line to implement changes. Depending on the system requirements, partial reconfiguration can be static or dynamic.

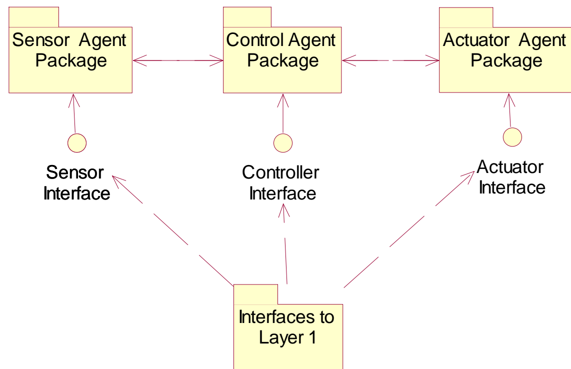

Fig. 2. UML Implementation of the Layer 1entities

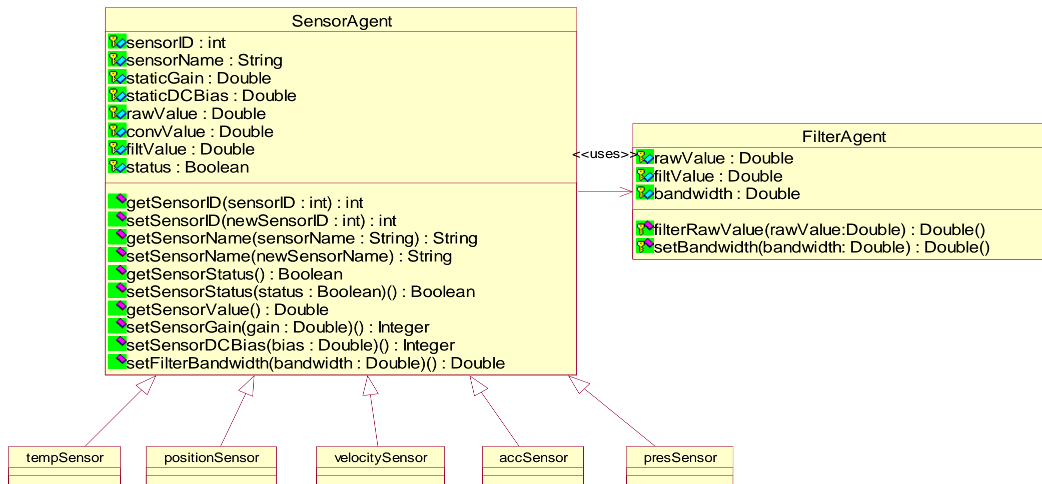

Fig. 3. Generalization of the sensor class

B. Software Implementation for Reconfigurable Computing

The implementation of the Layer 1 entities is shown in Fig. 2. The controller, sensing and actuation functions are abstracted out and implemented as separate packages with well defined interfaces. The controller communicates with the sensing module and the actuator module through the interface provided and is independent of the actual implementation within these modules. Providing standardized interfaces makes possible the implementation of distributed sensors and actuators. Further, by encapsulation of the sensing and actuation functions, system requirements such as plug-and-play of sensors, and fault accommodation can be achieved. Exposing the interface to these packages also aids in system diagnostics and testing of the low level functionality. The generalization of the sensor class is shown in Fig. 3.

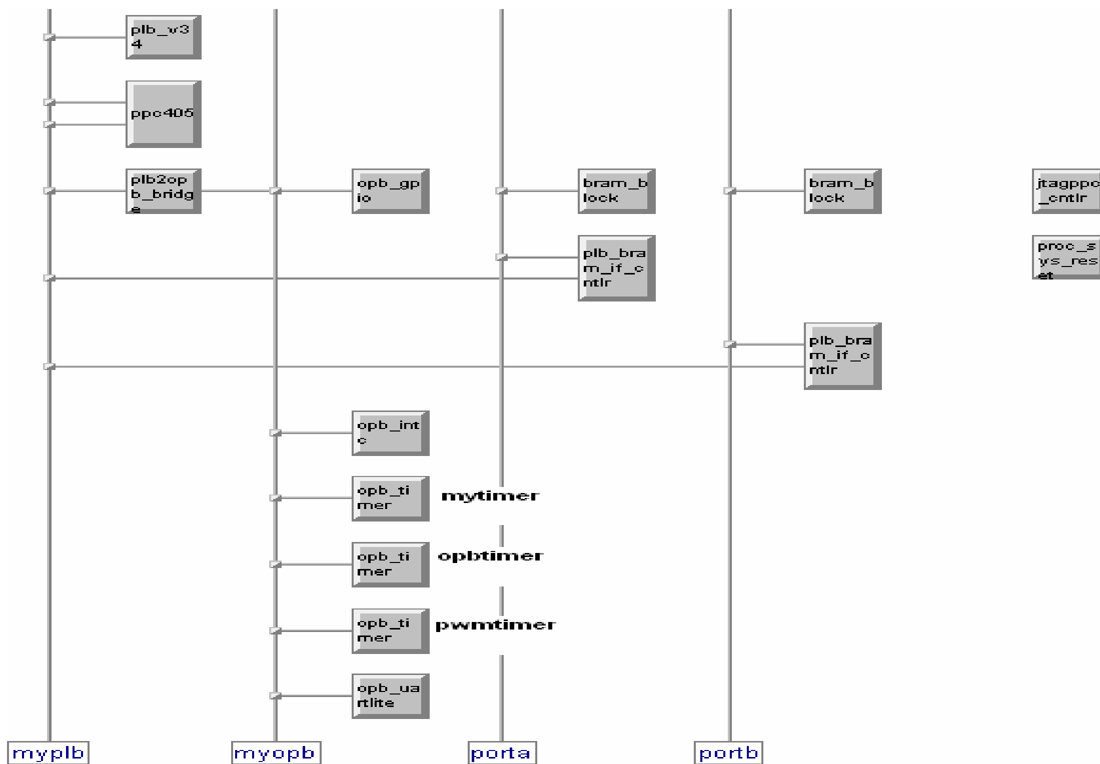

Fig. 4. Implementation of PWM motor control with dynamic reconfiguration for fault accommodation.

IV. CASE STUDY

The proposed framework is tested by implementing the L1 layer on the Xilinx Virtex-II Pro platform [16]. This platform was selected based on its capability in implementing reconfigurable architectures, and the excellent development tools and product support. The Virtex-II Pro XC2VP4 has a PowerPC core, 6768 logic cells, 504 KBits BRAM, 4 3.125 Gbps RocketIO transceivers, and 3.01 Mbits configuration space.

The Xilinx Virtex-II Pro device is a user programmable gate array with embedded PowerPC processor and embedded high-speed serial transceivers. The Xilinx Virtex architecture is coarse grained and consists of a number of basic cells called configurable logic blocks (CLBs). These logic blocks are arranged in rows and columns, with each CLB consisting of four logic cells arranged in two slices. Each CLB also contains logic that implements a four-input look up tables (LUTs) [13]. Each slice contains two function generators, two storage elements, arithmetic logic gates, large multiplexers, wide function capability, fast carry look ahead chain, and horizontal cascade chains. The function generators are configurable as four input look up tables (LUTs), sixteen bit shift registers, or as sixteen bit selective RAM memory. Each CLB also has fast interconnect and connects to a generalized routing matrix (GRM) to access general routing resources. The Virtex-II Pro has SelectIO-Ultra blocks (IOBs) that provide the interface between the package pins and the internal configurable logic. Active Interconnect Technology connects all these components together. The overall interconnection is hierarchical and is designed to support high speed designs [16].

The programmable elements in the Virtex-II Pro, including the routing resources, are controlled by values stored in the static memory cells. The device is configured by loading the bitstream into the internal configuration memory. These values can be reloaded to change the functions of the programmable elements. The Xilinx Virtex family of FPGAs supports both partial as well as dynamic reconfiguration. Partial reconfiguration can be achieved in one of the two ways, namely Module-based partial reconfiguration and difference-based reconfiguration. In the module-based reconfiguration, the entire module can be reconfigured. The height of the reconfigurable module is the height of the device and the module can cover one or more columns. In difference-based reconfiguration, the reconfiguration is done by making a small change in the design, and then generating a bit-stream based only on the differences in the two designs. Switching the configuration from one implementation to another is easy and very quick. The process of full reconfiguration is demonstrated in Fig. 4. The system is designed with PPC405 processor core, SDRAM controller connected to Processor Local Bus (PLB) and GPIO devices like Leds, Push buttons, UART and dip switches are connected to its On-chip Peripheral Bus (OPB). These are the components available on the board, so the first step is to verify the proper

functionality of all these components. To do this, the processor boots up with a configuration file to test all the components. On successful completion of built in self test, the processor fetches the second configuration file to configure itself and the board, switching into operational mode. If subsequent reconfiguration of the system requires additional components, say serial communication, then a different module can be generated with a UART device added to the OPB. Module based reconfiguration will then result in enhanced system capability. Since the reconfiguration can be done in real time while the system is operational, system components can be added in real time to address changing needs during the retasking of the system.

In the second design example, a PWM generator is implemented in the hardware to control the drive motors of the robot. Timer 1 (pwmTimer) is configured to generate the PWM signal while Timer 2 (opbTimer) is configured in the "capture" mode to sense the feedback signal. If a fault is detected during operation, then (a new) timer (myTimer) is activated and the output of this timer is switched to the output pins. The changes involved in the reconfiguration between the two designs are relatively small and small-bit manipulation is ideal for this type of reconfiguration. The control cycle in this example was executed in real time with a sampling rate of 20 msecs. The time for reconfiguration was of the order of a few micro-seconds showing that dynamic fault accommodation was achieved in real time.

V. Conclusions

In this paper, an architecture was presented that facilitates the implementation of intelligent embedded controllers. The system requirements were analyzed and the impact of these on the selection of hardware and software components was discussed. The design of reconfigurable hardware and software modules to implement "intelligence" at all levels of the system was developed. The application of these concepts to control problems was demonstrated through a case study.

Acknowledgment

The authors gratefully acknowledge the assistance of the U.S. Department of Defence, Army Research Office in supporting this work through grant # DAAD 19-03-1-0142.

References

[1] IEEE 1451.1, "Standard for smart transducer interface for sensors and actuators – Network-capable application processor (NCAP) information model," 1999.

[2] IEEE 1451.2, "Standard for a smart transducer interface for sensors and actuators – Transducer to microprocessor communication protocols and transducer electronic data sheet (TEDS) format," 1997.

[3] J. S. Albus, "Features of intelligence required by unmanned ground vehicles," *Proc. Performance Metrics for Intelligent Systems Workshop*, 2000.

[4] J. S. Albus and A. M. Meystel, *Engineering of Mind: An Introduction to the Science of Intelligent Systems*, Wiley Series on Intelligent systems, 2000.

[5] F.M. Proctor, B. Damazo, C. Yang, and S. Frechette, "Open architectures for control," *National Institute on Standards and Technology*, Internal report, NISTIR-5307, 1993.

[6] L. Wills, S. Kannan, S. Sander, M. Guler, B. Heck, J.V.R. Prasad, D. Schrage, and G. Vachtsevanos, "An open platform for reconfigurable control," *IEEE Control Systems Magazine*, pp. 49-64, Jun. 2001.

[7] R. P. Bonasso, R. J. Firby, Erann Gat, David Kortenkamp, David P. Miller, Mark G. Slack., "Experiences with an architecture for intelligent, reactive agents," *J. of Experimental and Theoretical Artificial Intelligence*, **9**(2-3): pp237-256, 1997.

[8] D. P. Schreckenghost, P. Bonasso, D. Kortenkamp, and D. Ryan, "Three tier architecture for controlling space life support systems", *Proc. IEEE Int. Joint Symposia on Intelligence and Systems*, pp. 195-201, 1998.

[9] T. Balch, and R. Arkin., "Behavior-based formation control for multirobot teams," *IEEE Trans. on Robotics and Automation*, **14**(6): pp 926-939, 1998.

[10] L. E. Parker, "ALLIANCE: An architecture for fault tolerant multi-robot cooperation," *IEEE Trans. on Robotics and Automation*, **14**(2):pp220-240, 1998.

[11] R. Simmons, T. Smith, M. B. Dias, D. Goldberg, D. Hershberger, A. Stentz, R. Zlot, "A layered architecture for coordination of mobile robots," in *Multi-Robot Systems: From Swarms to Intelligent Automata," Proc. from the 2002 NRL Workshop on Multi-Robot Systems*, A. Schultz and L. Parker (eds.), Kluwer, pp. 103-112, 2002.

[12] S. Commuri, J. Sarangapani, "Smart Embedded Systems for Control," *Workshop – IEEE Intl. Symposium on Intelligent Control*, Houston, Texas, 2003.

[13] S. Donti, and R. L. Haggard, "A survey of dynamically reconfigurable FPGA devices, " *Proc. IEEE.*, vol. 8, pp. 422-426, 2003.

[14] J. Harkin, T. M. McGinnity, and L.P. Maguire, "Partitioning methodology for dynamically reconfigurable embedded systems, "*IEE Proc. Comput. Digitial Technology*, vol. 147, no. 6, pp. 391-396, 1996.

[15] D. Mesquita, F. Moraes, J. Palma, L. Moller, N. Calazans, "Remote and partial reconfigurationof FPGAs: tools and trends," *Proc. Intl. Parallel and Distributed Processing Symposium*, 2003.

[16] Xilinx, Inc., *Virtex-II Pro: Platform FPGA Handbook*, UG012, v 2.0., 2002.