Efficient Controller Implementations for Robot Control

S. COMMURI, V. TADIGOTLA, L. SLIGER
Stephenson Research and Technology Center
101 David L. Boren Blvd, Room 1050
University of Oklahoma, Norman, OK-73019
UNITED STATES OF AMERICA

Abstract: - Field Programmable Gate Arrays (FPGAs) have emerged as the platform of choice for the rapid prototyping and testing of hardware circuits. The dynamic creation of hardware circuits in FPGAs provides an efficient mechanism for customizing the hardware for specific task objectives. The ability of these devices to be reprogrammed during run-time enables the hardware circuitry to adapt to changing task requirements and to accommodate system faults. In this paper, the use of FPGAs in implementing different behaviors and controllers for a mobile robot application is presented. It is shown that the computational hardware can be optimized for each task and sophisticated behaviors and controllers implemented in an efficient manner. The proposed approach is validated through a case study where a robot is configured to meet application specific requirements and the controllers on this robot are modified dynamically using Xilinx Virtex-II Pro FPGAs.

Key-Words: - FPGA, reconfiguration, fault tolerance, controllers for mobile robots.

1 Introduction

The complexity and the cost of conventional robot controllers have limited their use to routine, mundane tasks. Special purpose hardware and software deigned for experimental robots have increased the range of applications of the robots, but the inflexible nature of the solution is a restriction to their practical implementation. Recent advances in computational hardware and software allows for complex behaviors to be implemented in robots. These behaviors permit the robot to handle variations in the environment, and meet multiple task objectives in an efficient manner.

System On Chip (SoC) platforms based on conventional Application Specific Integrated Circuits (ASICs) cannot implement complex systems in an efficient and flexible manner [1]. Reconfigurable SoC's based on Field Programmable Gate Arrays (FPGAs) are, therefore, being designed to meet these requirements. This technology has been popularized in the recent past and there are a number of products based on high density FPGAs. In contrast to standard cells and gate arrays, FPGAs can be easily erased or reprogrammed [2], [3]. The latest version of these FPGAs introduce the concept of 'Dynamic Run-time Reconfiguration', where only a small portion of the circuitry is modified at run-time while the system remains functioning [4].

FPGAs have the advantages of being reconfigurable, having a reduced time to market and reduced one-time cost compared to their ASIC counterparts, but they are slower and use more power. Implementing controllers in reconfigurable hardware is one of the efficient and high-performance alternatives to the controllers implemented in conventional hardware [5], [6]. This is especially useful in robotics where the robot requires different control strategies for performing different tasks. The availability of low cost FPGA-based hardware [7] – [9] facilitates the implementation of intelligent robots where the hardware and software of individual robots can be modified in run-time to suit the needs of the application.While robots can attain limited functionality through the use of ASICs and special purpose microcontrollers, FPGA based implementations can be simultaneously optimized to meet the needs of different applications.

In this paper, the implementation of different controllers for specific tasks of a mobile robot is addressed using Xilinx Virtex II-Pro FPGA system. The configuration and selection of different sensor suites and controllers for robot navigation and for 'wall following' and 'leader following' tasks are demonstrated. The reconfiguration of the hardware circuitry is exploited to accommodate system faults and improve the robustness of the overall design. Case studies show that the proposed approach results in increased efficiency in the use of system resources and in systems that are robust to failures. The rest of the paper is organized as follows: A brief background on implementing reconfigurable

systems using Xilinx FPGAs is discussed in Section 2. The design and implementation of reconfigurable controllers for robot applications is discussed in Section 3. The advantages of the proposed design are demonstrated through a practical implementation of an intelligent robot in Section 4. The summary of the results of the research are summarized in Section 5 and the conclusions are presented in Section 6.

2 Reconfiguration in FPGA Based Systems

FPGAs provide an array of gates that can be configured to perform a logic function. The interconnection of these functional modules to execute a computational task is accomplished by loading a configuration file onto the FPGA. While in the past the FPGAs supported only static configurations, recent FPGAs support dynamic reconfiguration where the functionality of the FPGA can be changed during runtime. Further, these devices allow partial reconfiguration where a portion of the device can be reprogrammed during runtime. In this case, Dynamic Partial Reconfiguration (DPR) is done while the device is active, i.e. certain areas of the device are reconfigured while the other areas remain operational and are not affected by reconfiguration. Virtex FPGAs of Xilinx family support partial reconfiguration and are used to demonstrate the concepts presented in this paper. The following discussion is an abridged version of the FPGA background presented in [7].

The Virtex-II Pro device is a user programmable gate array with embedded PowerPC processor [10], [11]. The key-component of the Virtex-II Pro is the Configuration Logic Block (CLB). 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). Reconfiguration in Xilinx Virtex FPGA involves erasing the contents of the CLBs and configuring them with new logic. *Full* Reconfiguration is accomplished by erasing all the CLBs and configuring them with new logic elements. *Partial* Reconfiguration, on the other hand, replaces only the CLBs specified by the bitstream while the remaining CLBs retain their current configuration and remain functioning during the reconfiguration.

Reconfiguration in such systems can be accomplished using two different methods: Difference-based flow and Module based flow [10], [11]. *Difference-based flow* is suited for those applications requiring a minor change in the hardware configuration. System reconfiguration can be achieved by downloading a partial bitstream representing the difference onto the FPGA. This bitstream is utilized to change only the configuration of the affected CLBs on the platform while the remaining CLBs continue to function normally. During *Module-based flow*, the full design is partitioned into modules, some of which can be static while the others are reconfigurable. These modules communicate with each other through Bus Macros and individual reconfigurable modules can be reconfigured without affecting the overall system.

A detailed description of designing partial and dynamically reconfigurable applications on Virtex-II FPGAs can be found in [12] – [15], [17] - [21]. In the next section, the application of FPGAs in the implementation of controllers for different robot tasks is presented.

3. Experimental Test Bed and Design of the Controller

In this section, the implementation of dynamically reconfigurable controllers for different robot tasks is presented. The prototype testbed consists of a four wheeled *Tamiya Xtreme* Truck with 1/10 scale monster truck chassis and is shown in Fig 1. This mobile base is retrofitted with Transmissive Optical Encoders (EM1 & HEDS-9100-G00 manufactured by US Digital), IR Range sensors (SHARP GP2D02), low power multi-channel wireless transceivers (RX-Wi 232 DTS manufactured by Radiotronix, Inc.), Vision based on CMUCam (CMUCam1 capable of transmitting 17 frames per second) and high torque servos for steering (HiTec HS-945MG). The robot controller uses a Virtex-II Pro FPGA board (Memec V2PFF672) for reconfiguring the controllers.

Fig 1. Prototype Testbed used for the robot

The Virtex-II Pro FPGA platforms have an embedded IBM PowerPC 405 processor connected to the Peripheral Logic Bus (PLB) (Fig 2). The PLB is connected to the OnChip Peripheral Bus (OPB) through PLB/OPB Bridge. All the Static and

Reconfigurable modules (controllers) are connected to the OPB. Modules like Wireless Transceiver Interface Module, Motor Drive Interface Module, Wheel Encoder Interface Module, SRAM Controller etc., are the Static Modules as they are present in every configuration of the robot.

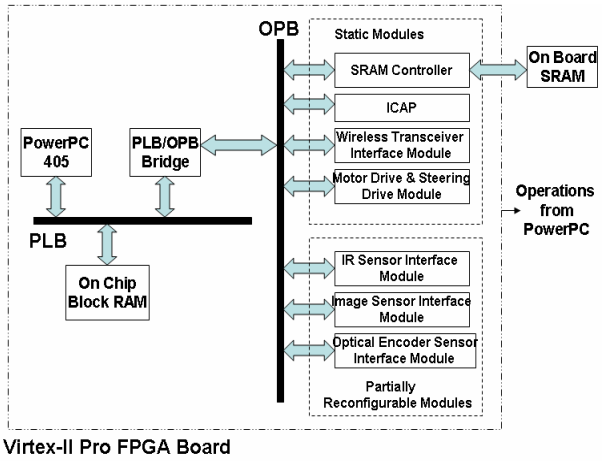

Fig 2. Internal Architecture of the Robot Controller

The Reconfigurable Modules shown in Fig 2 are the controllers required by the behaviors to interface to the necessary hardware on the testbed. For example, in the 'Leader/Follower' mode the robot utilizes a CMUCam while in the 'Wall-Following' mode the control is based on the input from IR Sensors. When the system is running, changing task requirements will lead to the selection of appropriate behaviors. These behaviors are executed under software control and result in the selection of appropriate modules and drivers. This process is controlled by the PowerPC processor by the downloading of an appropriate partial bitstream through the ICAP interface.

3.1 Implementation of Controllers

In a typical application, different tasks require different navigational capabilities of the robot. For example, a simple navigation command could result the robot to attain a specified heading and forward velocity. On the other hand, autonomous behaviors would require the robot to navigate along a wall or follow a beacon or a leader. In either case, changing navigational requirements require the adoption of different control logic and the use of appropriate sensors. In this section, we explore the implementation of the controller for two such scenarios, namely 'wall following' mode and 'leader following' mode.

3.1.1 Controller for following the wall

In the task where the robot is required to navigate along a wall, the control problem can be decoupled and the heading control treated independent of the velocity control for the robot. The robot is required to move forward at a constant velocity unless this behavior is modified by the need to avoid obstacles. The heading control in this case is specified as

$$u_h = 90^0 + K_h(d_f - d_r) + K_d(d - \frac{d_f + d_r}{2}) \quad (1)$$

where d_f, d_r are the distance of the front and the rear wheels from the wall. $(d_f - d_r)$ represents the deviation from the line parallel to the wall, d is the desired spacing, and K_h, K_d are the feedback gains. It can be easily seen that the control strategy ensures proper wall following by correcting errors in heading $(d_f - d_r)$ and the error in the average distance from the wall $\left(d - \frac{d_f + d_r}{2}\right)$ (Fig 3).

3.1.2 Controller for following a leader:

A simple leader-follower configuration of two autonomous robots is shown in the Fig 4. Here $\mathbf{R_1}$ is the lead robot with the control input vector $u_1 = [u_{d1} \;\; u_{h1}]^T$, and $\mathbf{R_2}$ is the follower robot with the control input vector $u_2 = [u_{d2} \;\; u_{h2}]^T$. The desired separation between the two robots is d_{12}^d and the desired relative bearing is φ_{12}^d. The current separation between the robots d_{12} and the current relative bearing φ_{12} can be found based on the visual tag as seen in the follower camera image plane.

$$\begin{aligned} \dot{\varphi}_{12} &= k_{b1} + k_{b2}(x - x') \\ d_{12} &= k_{d1} + k_{d2}(N) \end{aligned} \quad (2)$$

Where $x = \frac{x_1 + x_2}{2}$ and $x' = \frac{x'_1 + x'_2}{2}$ (as seen in the image plane Fig 4) and N is the number of pixels of the beacon as seen in the camera image plane. k_{b1}, k_{b2}, k_{d1}, and k_{d2} are the gains to be tuned to estimate the calculated distance and heading values.

4. Case Study

In this case study, the advantages of partial reconfiguration are demonstrated through a practical implementation of an intelligent robot. Basic navigation is accomplished through the use of the wheel encoders and a PID controller. This behavior is modified by an obstacle avoidance scheme where

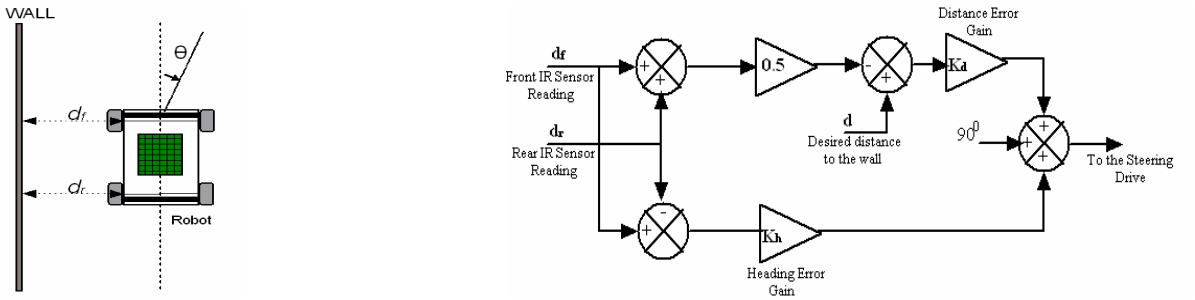

Fig 3. Block Diagram of the Controller for Wall Following

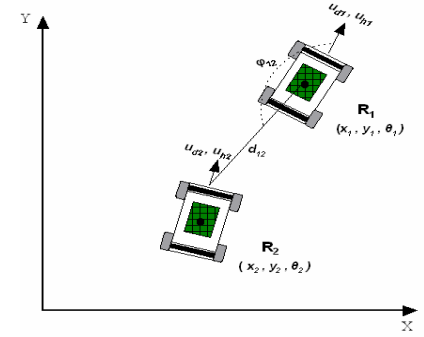

Fig 4(a). Leader/Follower Configuration

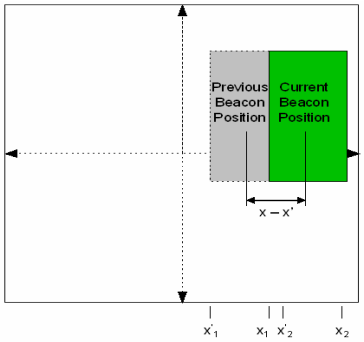

Fig 4(b). Image of Beacon in the Camera Frame

an obstacle is detected using infrared proximity sensors or the CMU CAM. Using these basic building blocks, sophisticated behaviors like Wall Following and Leader-following are implemented. Two IR sensors (SHARP GP2D02) on the side of the mobile robot are used to determine the distance from the wall as well as the heading of the robot. These sensors have a range between 10cm (~4") to 80cm (~30") and an accuracy of ± 2 cms (~1 inches) when the sensors are stationary at a distance of 40cms (~16 inches) from the wall.

In this case study, Autonomous fault handling and self-reconfiguration [22] are used to demonstrate the dynamic run-time implementation of controllers in a robot. Here, the robot is initialized and a task is assigned to it. Specifically, the robot is required to move forward a distance of 2 m (~6 feet) while maintaining a distance of 0.35 m (~14 inches) from a straight wall. In case of an error with the sensors, then the on-board fault handling algorithm checks available resources and reconfigures the appropriate sensors and drivers to complete the task. In the example shown (Fig 5), the IR sensors fault out 9 seconds after task initiation, rendering the robot incapable of completing the task. The on-board fault handling algorithm then initiates a partial reconfiguration of the FPGA to configure a CMUCAM. The robot then continues the task using a beacon to maintain the desired heading. The screen shot of the Partial Reconfigurable Module, Static Modules and Bus Macros within the FPGA is shown in Fig 6. The Partial Reconfigurable Module (PR Module) communicates with the Static Modules using the Bus Macros. This designed occupied 59% of the FPGA and was developed using Xilinx Early-Access Partial Reconfiguration Design Flow method (EA PR Design Flow). This methodology is expected to reduce the complexity of the Partial Reconfiguration by 30% [5]. One important feature in EA PR design flow is that, it allows for Partial Reconfiguration regions of any rectangular size allowing the efficient use of the FPGA. It can be seen in Fig 6 that the Static Module can occupy the same columns as the Reconfigurable Module. The EA PR flow also allows signals in the base design to cross through a partially reconfigurable region without the use of a Bus Macro. This improves the timing performance and simplifies the process of building a partially reconfigurable design.

5 Summary

In this paper, the development of intelligent robots using reconfigurable hardware and software was presented. In the implementation, the static module was designed to include the PowerPC processor core, ICAP interface, Bus Macros, and the OPB and PLB buses. Modules requiring dynamic modification in real-time are implemented as reconfigurable modules and communicate with the rest of the hardware through the Bus Macros. The screen shot of the FPGA_Editor showing the FPGA containing Partially Reconfigurable Module, Static Module and Bus Macros is given in Fig 6. This design is modular and incurs minimal hardware and computational overhead for each operational mode of the system.

The design in the case studies was developed

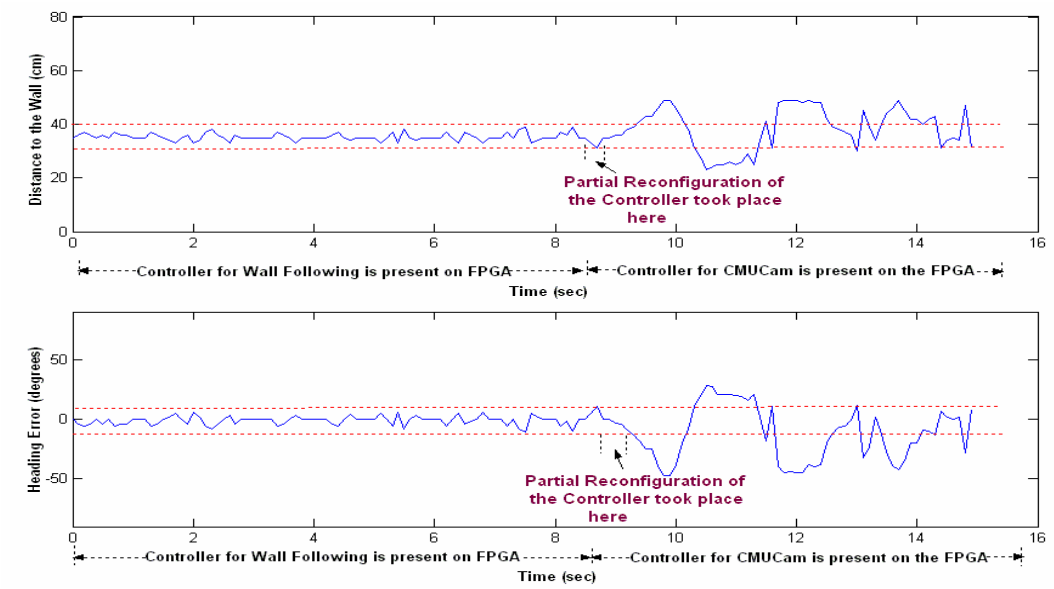

Fig 5. Plot of the data obtained from the robot depicting dynamic implementation of self-reconfigurable controllers

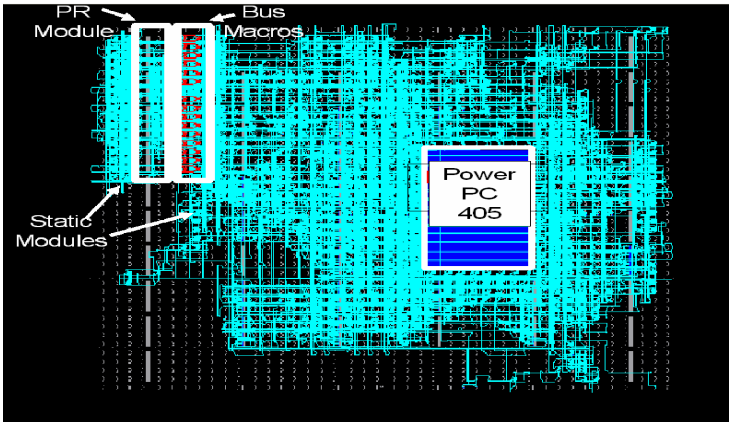

Fig 6. Screen shot of the FPGA_Editor showing PR Modules and Static Modules

using Xilinx Early-Access Partial Reconfiguration Design Flow method (EA PR Design Flow). In contrast with earlier methods where there were restriction on the size and location of the reconfigurable modules, this design method allows the assignment of any rectangular region for the reconfigurable modules. This minimizes unused FPGA space and improves resource utilization. The EA PR flow also allows signals in the base design to cross through a partially reconfigurable region without the use of a Bus Macro. This improves the timing performance and simplifies the process of building a PR design. The basic navigation, wall following, and leader following modules are implemented in the partially reconfigurable modules. In the case study discussed above, static modules occupy 58.1% of the available FPGA space while the 'wall follower' behavior module and the 'leader follower' behavior modules occupy 2.4% and 8.4% of the FPGA space respectively. Implementing all these modules in the static portion of the FPGA without partial reconfiguration would require 71.2 % of the FPGA space with the additional software overheads. As the number of behavior modules on the robot increases, the slice count increases. So the FPGA is restricted to only a few behaviors when partial dynamic reconfiguration is not used. This is a serious proble if a single robot is to be used for executing numerous tasks.

The problem discussed above can be addressed by implementing the same design in the FPGA using partial dynamic reconfiguration. In this case, the overall slice count is the sum of slice count of the static portion and the slice count of the partial reconfigurable module. The size of the PR Module should be at least of the largest behavior module size. In our design we allocated 9% of the FPGA for the PR Module which is sufficient to accommodate any behavior. Using partial reconfigurable FPGAs, all the three behavior modules can be accommodated within 65% of the FPGA space. The bitstream of all the behavior modules are stored in an external

memory and the appropriate behavior is accommodated into the PR Module as of when required. So as the number of the behavior modules increases, the slice count of the FPGA doesn't vary much. This gives the flexibility of using a singlrobot for performing various tasks without increasing the slice count on the FPGA.

The average resulting bit-stream for each of the configurations is 1175 kBytes resulting in a reconfiguration time of 23.4 milli-seconds and a worst net delay of about 9.686 nano-seconds. From these results, it can be seen that the reconfiguration can be achieved within the duration of a standard control cycle. While these results demonstrate the implementation for two simple behaviors in the robots, the FPGA utilization and efficiency becomes significant as the number of required behaviors in a robot increase.

6. Conclusions

In this paper, the run-time reconfiguration of FPGA based robot controllers was presented. For the first time, the dynamic reconfiguration of FPGAs was harnessed to extend the capability of a robot. Fault accommodation and behavior reassignment of robots was addressed using the concept of hardware and software reconfiguration. In our future work, the reconfiguration will be extended to embed intelligence and control algorithms into the robot architecture.

References

[1] K. Compton, S. Hauck, Reconfigurable Computing: A Survey of Systems and Software, *ACM Computing Surveys*, Vol. 34, No. 2, 2002, pp 171-210.

[2] G. Goslin, A guide to using field programmable gate arrays (FPGAs) for application-specific digital signal processing performance, *Tech. Rep.*, Xilinx Inc., San Jose, 1995.

[3] D.Buell, J.Arnold, and W.Kleinfelder, *Splash 2. FPGAs in a Custom Computing Machine.* New York: IEEE Press May 1996.

[4] Michael Barr, A Reconfigurable Computing Primer, *Multimedia Systems Design*, Sep. 1998, pp. 44-47.

[5] M. Bednara, K. Danne, M. Deppe, O.Oberschelp, F. Slomka, and J. Teich, Design and Implementation of digital linear control systems on reconfigurable hardware, *EURASIP Journal on Applied Signal Processing,* 2003, pp: 102-127.

[6] R. Kasper and T. Reinemann, Gate level implementation of high speed controllers and filters for mechatronic systems, *Mechatronic Workshop ,* 2000.

[7] *Virtex – II platform FPGA user guide*, version 1.8, Xilinx Inc., 2005.

[8] www.xilinx.com

[9] www.altera.com

[10] Virtex Series Configuration Architecture User Guide, *Xilinx Application Note XAPP151*, version 1.1, Xilinx, Inc. ,1999.

[11] *Virtex –II Pro Platform FPGA User Guide*, version 1.8, Xilinx, Inc ., 2004.

[12] Two Flows for Partial Reconfiguration: Module Based or Difference Based, *Xilinx Application Note XAPP 290*, version 1.2, Xilinx Inc., 2004.

[13] C. Bobda, B. Blodget, M. Huebner, A. Niyonkuru, A. Ahmadinia, M. Majer, Designing Partial and Dynamic Reconfigurable Applications on Xilinx Virtex-II Pro FPGAs using Handel-C, *Technical Report,* University of Erlangen-Nuremberg, Germany, Nov. 2004.

[14] H. Tan, R.F. DeMara, A.J. Thakkar, A. Ejnioui, J.D. Sattler, Complexity and Performance Tradeoffs with FPGA Partial Reconfiguration Interfaces, submitted to *13th Reconfigurable Architectures Workshop(RAW'06),* Greece., April-2006

[15] Klaus Danne, Christopher Bobda, and Heiko Kalte, Run-Time Exchange of Mechatronic Controllers using Partial Hardware Reconfiguration, *Lecture Notes in Computer Science (LNCS) 2778,* FPL-2003, pp: 272-281.

[16] www.ibm.com/rational

[17] Yan Meng, A Dynamic Self-Reconfigurable Mobile Robot Navigation System, *Proceedings of IEEE/ASME International Conference on Advanced Intelligent Mechatronics,* July 2005, pp: 1541-1546

[18] Fabrizio Ferrandi, Macro D. Saantambrogio, Donatella Sciuto, A design methodology for dynamic reconfiguration: the CARONTE Architecture, *Proceedings of 19th IEEE International Symposium on Parallel and Distributed Processing,* April 2005, pages: 4.

[19] Sergio Lopez-Buedo, Javier Grrido and Eduardo I. Boemo, Dynamically Inserting, Operating and Eliminating Thermal Sensors of FPGA-based systems, *IEEE Transactions on Components and Packaging Technologies",* Vol. 25, Decemeber-2002, pp.561-566.

[20] Gregory Mermoud, A Module-Based Dynamic reconfiguration tutorial, *Technical Report,* Logic Systems Laboratory, Ecole Polytechnique Federale de Lausanne, November 2004.

[21] Brandon Blodget, Scott McMillan, A lightweight approach for embedded reconfiguration of FPGAs, *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'03),* Vol. 3, 2003, pp: 1530-1531.

[22] V. Tadigotla, L. Sliger, S. Commuri, FPGA based dynamic Run-Time behavior reconfiguration of robots, *submitted to IEEE International Symposium on Intelligent Control,* Oct 2006.