J Intell Robot Syst
DOI 10.1007/s10846-007-9131-3

Task-based Hardware Reconfiguration in Mobile Robots Using FPGAs

Sesh Commuri · V. Tadigotla · L. Sliger

Received: 27 June 2006 / Accepted: 11 January 2007

Abstract This paper presents a methodology for the realization of intelligent, task-based reconfiguration of the computational hardware for mobile robot applications. Task requirements are first partitioned into requirements on the system hardware and software. Architecture is proposed that enables these requirements to be addressed through appropriate hardware and software components. Hardware–software co-design and hardware reconfiguration are utilized to design robotic systems that are fault-tolerant and have improved reliability. It is shown that this design enables the implementation of efficient controllers for each task of the robot thereby permitting better operational efficiency using fixed computational resources. The approach is validated through case studies where a team of robots is configured and the behavior of the robots is dynamically modified at run-time. It is demonstrated through this implementation that the design procedure results in increased flexibility in configuration at run-time. The ability to reconfigure the resources also aids collaboration between robots, and results in improved performance and fault tolerance.

Key words autonomous ground vehicles · fault-tolerance · FPGA · hardware reconfiguration · intelligent robots

1 Introduction

The design of intelligent mobile robot teams is the next frontier in robotics research. Mobile robots are now being implemented in a variety of applications from security patrols to space exploration. While the use of mobile robots has been successfully demonstrated, their adoption has not been widespread on account of their complexity and cost. Most applications require a team of robots working cooperatively to accomplish the mission objectives. Further, these applications require a high degree of fault tolerance and the ability

S. Commuri (✉) · V. Tadigotla · L. Sliger
School of Electrical and Computer Engineering, University of Oklahoma, Norman, OK, USA
e-mail: scommuri@ou.edu

to function under varying operating conditions. Design of such robotic systems cannot be accomplished by the simple integration of smart controllers or transducers [21, 22]. 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 [31, 46].

In the conventional design process, the required system functionality and performance targets are used to allocate computational resources. 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. This leads to a trade-off between the system performance and its cost and flexibility [31, 46]. One of the early approaches to intelligent control focused on the development of suitable hierarchical architectures for system autonomy [2]. Mathematical approaches to the analysis of intelligent systems were researched by Valavanis [43]. Approaches based on centralized planning [9, 33] as well as on distributed controls [5, 23, 29] were also attempted to address the requirements of efficiency and fault tolerance. As real time requirements for intelligent embedded systems grew more stringent, researchers considered a combination of both centralized and distributed architectures in their design [37]. Behavior based control was also studied to reduce overall computational demands and to make task-handling robust. Learning to coordinate the behaviors can lead to system robustness and to coordinating multiple behaviors and goals [24]. However, the centralized coordination approaches are difficult to apply to large scale systems with multiple behaviors and tasks [19, 32]. The scheduling of tasks and the communication system for supporting real-time message exchange in distributed architectures was addressed in Franchino, Buttazzo, and Facchinetti, [16]. Huntsberger et al. [20] propose control architecture (CAMPOUT) for cooperative robots performing tightly coupled tasks. Using this architecture, the mobility and manipulation of multiple robots was demonstrated. It was also shown that this architecture allows for autonomous adaptation to uncertainties in the environment. The cognition oriented approach [1] on the other hand, uses a situation-operator model to formalize the changes in the environment as a sequence of scenes and actions. This formalization is then used to implement the cognition-based control for the system.

In spite of above mentioned advances, low-cost flexible mobile robots are still a distant reality primarily due to the hardware and software issues in the design and the difficulty in their integration. The performance improvements that can be realized through the partitioning of applications in FPGA hardware are discussed in Galanis, Dimitroulakos and Goutis [17]. The use of reconfigurable hardware components, like Field Programmable Gate Arrays (FPGAs), for hardware and software reconfiguration has been discussed in a number of recent publications [10, 14, 18, 25, 27]. In this paper, an alternate approach to the design of intelligent mobile robots is addressed through the systematic partitioning of the overall system. The functionalities that require changing execution paths or those that impact performance are assigned resources that allow both hardware and software reconfiguration. A hierarchical architecture that allows for plug-and-play and fault tolerance at the lowest level and for learning and adaptive behaviors at the highest level is proposed where both the real-time performance and system intelligence can be addressed. 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 mobile robots [11, 28, 38, 44]. The proposed design methodology is demonstrated through the design of low cost robotic platforms that are easily configured at run-time, are capable of intelligent fault handling, and can collaborate and attain higher levels of functionality.

The rest of the paper is organized as follows. Section 2 presents the architectural requirements and a hierarchical architecture for intelligent mobile robots. The hardware and software requirements for such systems are discussed in Section 3. The experimental setup is presented in Section 4 and the experimental results in the implementation of reconfigurable robots are presented in Section 5. The major contributions of the paper and the future research directions are presented in Section 6.

2 Architectural Requirements for Intelligent Mobile Robots

An intelligent mobile robot must be capable of sensing the environment, have situational awareness based on the world models and the perceived information, and be able to choose appropriate behaviors that maximize the success of the task objectives. The architectures of such systems must address issues such as:

- The ability to dynamically configure and retask individual robots,
- The ability of the robots to identify and accommodate system faults,
- Distributed /centralized control of the robot teams,
- Information and resource sharing between multiple robots, and
- The efficiency of fault tolerance and learning algorithms.

The design must also address real-time and non real-time issues in sensor fusion, communication between network nodes, and the automation of the decision making process. These issues will be addressed using the architecture shown in Fig. 1. The design methodology developed in this paper is motivated by the 4D/RCS architecture [3] and enables the design of simple components whose performance can be rigorously analyzed. This architecture has evolved from the works of Passino [2] and the researchers at the

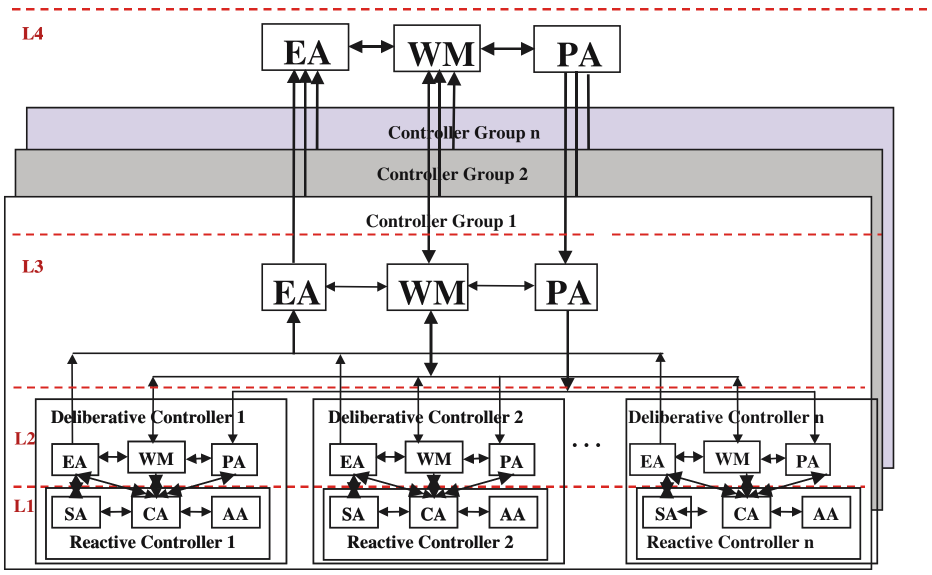

Fig. 1 Architecture for intelligent reconfigurable mobile robots

National Institute of Standards (NIST) in Gaithesburg, Maryland. Complex hierarchical systems can them be constructed using these low-level building blocks. This architecture is hierarchical in nature and allows system intelligence to be incorporated at all levels of the hierarchy.

The architecture in Fig. 1 envisions one or more robotic agents working as a group. At the lowest level (L1), each robot agent has 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 level (L1) is characterized by stringent real-time requirements and deterministic behaviors. At a very fundamental level, this design is adequate for a robotic agent to function and perform repetitive tasks in a structured environment.

In the hierarchical model shown in Fig. 1, the layer L2 is designed to meet the requirements of fault tolerance, uncertainty in the system model and the environment. 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). 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. While L1 is characterized by reactive control loops driven by tight real-time requirements, L2 incorporates elements that instill "intelligence" in the robot and is characterized by increased autonomy and less stringent real-time requirements.

A team of robots may consist of a number of individual robots possibly with differing sensor/actuator suites and capabilities. Tasking of individual robots and coordination between robots in a team are managed by the PA entity at the level of the robot 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. Team-level sensor fusion amongst the different robotic agents 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 robot agents in L2. The planning agent (PA) in this layer does the task decomposition from the mission requirements and updates the individual PAs 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 architecture described above is flexible and is not dependent on the type of controllers or algorithms implemented in any given layer. The hardware and software design that implements this architecture is now described.

3 Hardware and Software Requirements

In this section, the features of the embedded controllers that are suited for use in the architecture proposed in Section 2 are discussed. Traditionally, embedded controllers were implemented as specialized algorithms in 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. Application Specific ICs (ASICs) offer high efficiency in the implementation but lack the flexibility required by many applications. Software based reconfiguration provides the required flexibility in the implementation but is limited by the capability of the hardware and could also lead to excessive computational overheads. The objectives of efficiency and flexibility can be simultaneously met using Field Programmable Gate Arrays (FPGAs). Hardware–software co-design methods can then be used to design systems that are not only flexible but are highly efficient. The requirements for such a design are elaborated in this section.

3.1 Hardware Requirements for Reconfigurable Computing

Reconfigurable Computing is the ability of the system to reach through the hardware layer and change the data path for execution. FPGA-based reconfigurable hardware facilitates the implementation of intelligent robots where the configuration of individual robots can be modified to suit the needs of the application and the behavior of the robot team can be modified to accomplish overall mission objectives. In order to harness the power of hardware reconfiguration, the system should be partitioned into tasks executable in software and hardware. The tasks that require high levels of computations and stringent real-time operation can be implemented in hardware while tasks for general purpose computing can be implemented in software. The types of reconfigurations possible with these devices and the partitioning of the tasks for co-design are now discussed.

3.1.1 Full Vs Partial Reconfiguration

Built in 'Self Test' is the first task executed on system startup to verify proper functioning of the hardware. On successful completion of the startup sequence, the system transitions into an operational mode. Traditionally, the test sequences that could be run were limited by the functionality of the hardware. This limitation can be overcome by harnessing the reconfiguration power 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 improved. 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 sensing 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 remains in operation. Partial Reconfiguration is useful in robotic applications where the sensing/actuation needs and task requirements change in a dynamic fashion.

3.1.2 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.

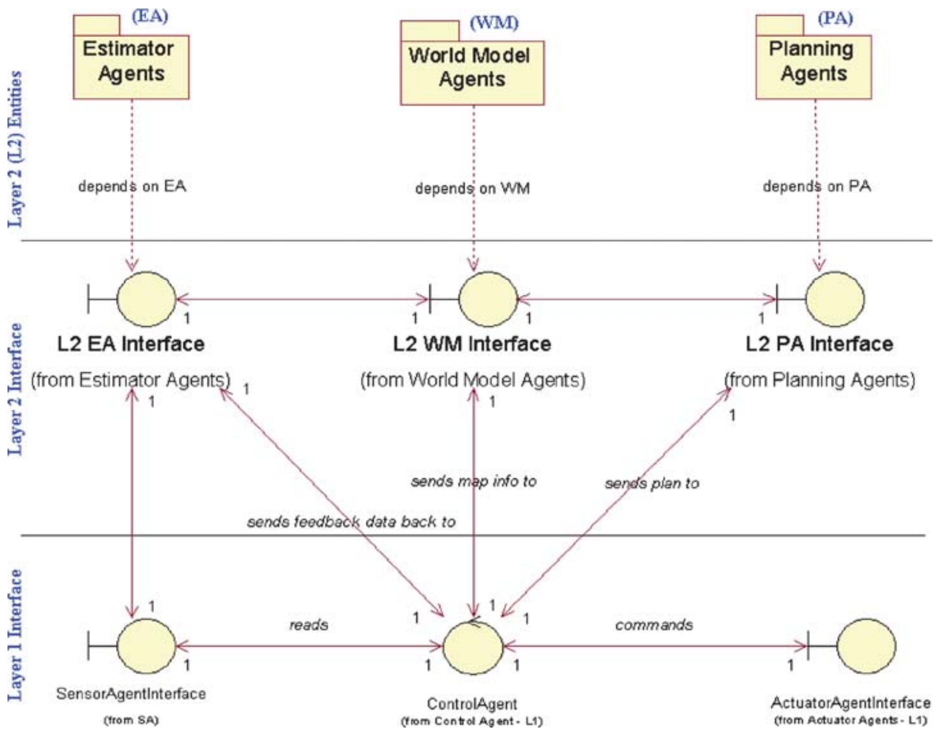

Fig. 2 Logical diagram of Layer 2 entities and interfaces to Layer 1

3.2 Software Architecture and Co-design

Reconfigurable hardware enables the implementation of optimized hardware components for different tasks of the robot. For this approach to be successful, the associated software components have to be designed simultaneously so that the changing execution paths in hardware are supported by the required software functionality. In order to achieve this, each layer in the architecture in Section 2 is packaged separately with well defined interfaces that enable interlayer communications and data exchange. For example, the interactions between the L1 and L2 entities are through well defined interfaces as shown in Fig. 2. Such an abstraction will allow for both centralized as well as distributed implementation of the controller software and is independent of the actual implementation. The desired functionality is depicted as a ‘Use Case’ model in Fig. 3.

4 Experimental Setup

The architecture detailed in Sections 2 and 3 will be implemented in a team of robots and the ability of each robot to be dynamically configured, tasked, as well as to coordinate with other robots will be demonstrated. In this section, the experimental setup for the implementation is first described. A mathematical model of the robot is developed and the control strategy is then formulated.

4.1 Robot Platform

The robot platform used for the experimental validation of the design concepts is built using the 'Tamiya Xtreme' radio control truck with 1/10 scale monster truck chassis. This truck has high performance servos for independent steering and all wheel drive (Fig. 4). The dynamics of this robot are given as follows.

$$\dot{x} = u_d \cos\phi; \quad \dot{y} = u_d \sin\phi; \quad \dot{\phi} = \omega_i \tag{1}$$

where (x, y) is the position of the robot in an arbitrary frame of reference and ω_i is the desired angular velocity (Fig. 5). $u = [u_d \quad u_h]^T$ represents the control input to the robot and comprises of the desired velocity (u_d) and the steering command (u_h). The range for the steering angle θ is $0° < \theta < 180°$, with 90° representing the straight ahead position. $\theta < 90°$ corresponds to the robot steering left and $\theta > 90°$ corresponds to robot steering right. It can be seen from Eq. 1 that the heading angle, φ, of the robot is obtained by integrating the steering angle along the trajectory of the robot.

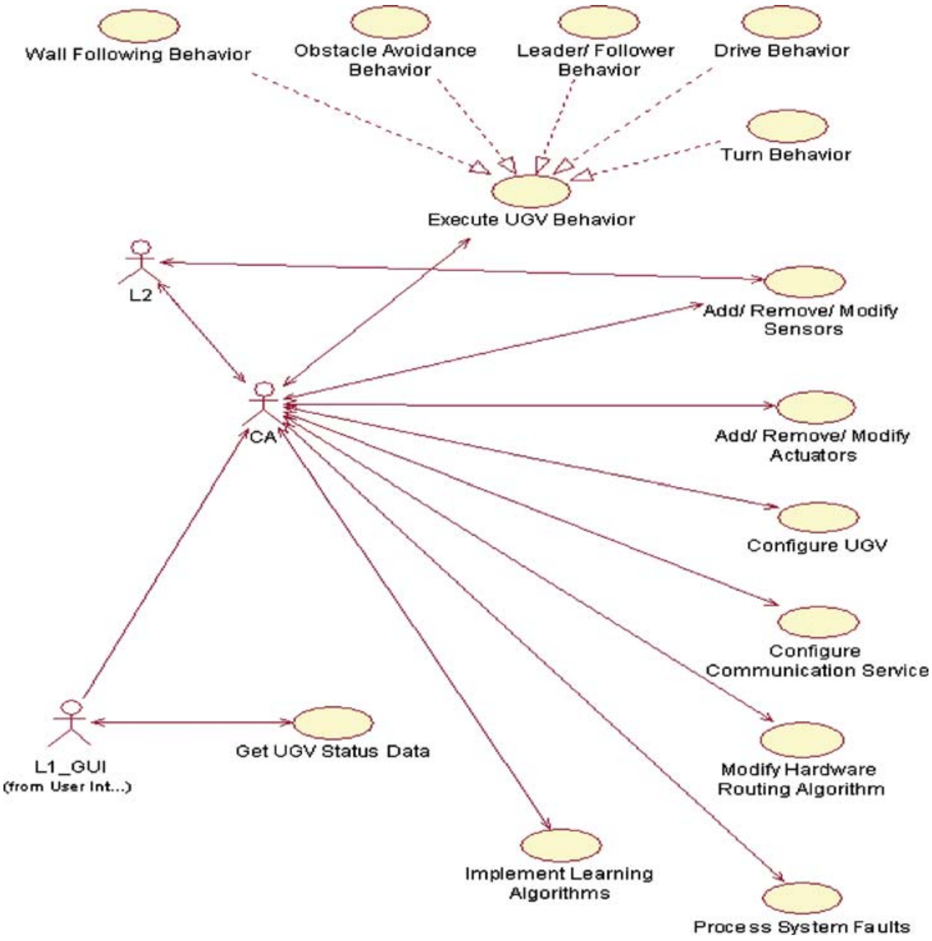
Wall Following Behavior
Obstacle Avoidance Behavior
Leader/ Follower Behavior
Drive Behavior
Turn Behavior
Execute UGV Behavior
L2
Add/ Remove/ Modify Sensors
CA
Add/ Remove/ Modify Actuators
Configure UGV
Configure Communication Service
L1_GUI
(from User Int...)
Get UGV Status Data
Modify Hardware Routing Algorithm
Implement Learning Algorithms
Process System Faults

Fig. 3 Use case view of Layer 1 operations

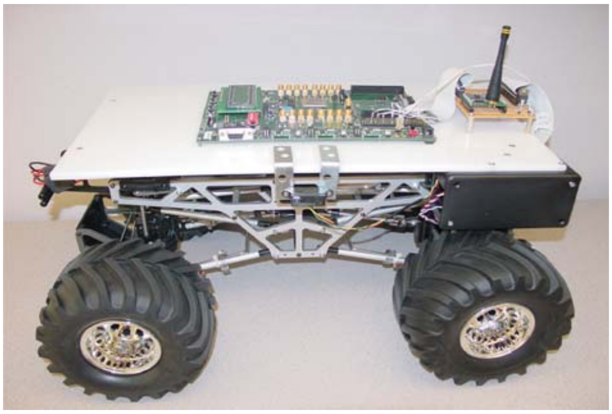
Fig. 4 Prototype mobile robot testbed with Virtex II pro reconfigurable platform

a. *Controller for Wall Following*: In a task where the robot is required to navigate along a wall, the control problem can be decoupled and the steering control treated independent of the velocity control for the robot. The robot is usually required to move forward at a constant velocity unless this behavior is modified by the need to avoid obstacles or other task requirements. The steering control in this case is specified as (Fig. 6):

$$u_h = 90^\circ + K_s\left(K_h\left(d_f - d_r\right) + K_d\left(\frac{d_f + d_r}{2} - d\right)\right) \tag{2}$$

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. $K_s(=\pm 1)$ is a constant that represents the location of the robot with respect to the wall and ensures that the robot steers away from the wall whenever $\frac{d_f+d_r}{2} > d$. 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 $\left(\frac{d_f+d_r}{2} - d\right)$.

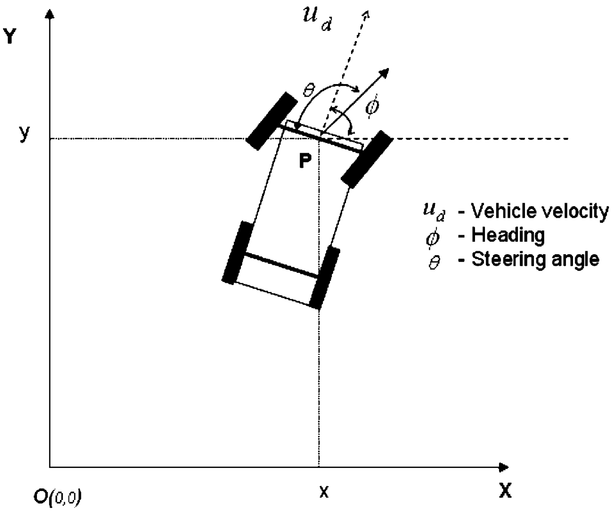

Fig. 5 Kinematic representation of the robot

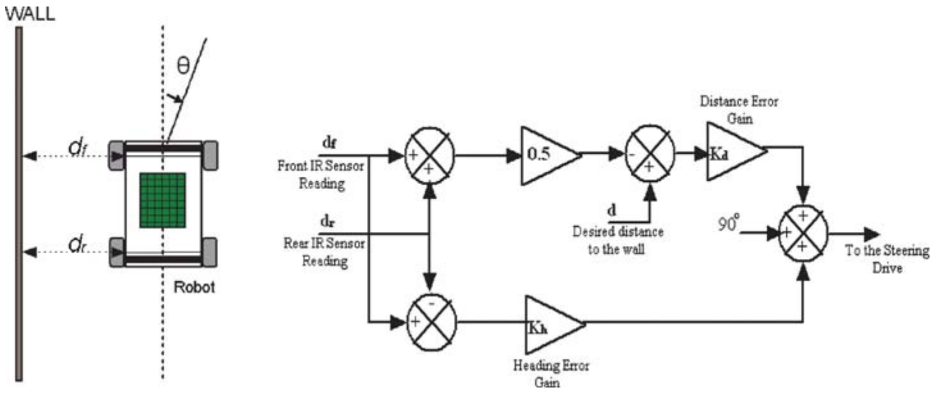

Fig. 6 Block diagram of the controller for wall following

b. *Controller for Leader-following*: A simple leader–follower configuration of two autonomous robots is shown in the Fig. 7. Here R_1 is the lead robot with the control input vector $u_1 = [u_{d1}\ u_{h1}]^T$ and R_2 is the follower robot with the control input vector $u_2 = [u_{d2} \quad u_{h2}]^T$.

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.

$$\begin{aligned} d_{12} &= k_{d1} + \frac{k_{d2}}{N} \\ \varphi_{12} &= \frac{1}{d_{12}}\left(k_{b1} + k_{b2}(x - x')\right) - \phi_1 \end{aligned} \tag{3}$$

where $x = \frac{x_1+x_2}{2}$ and $x' = \frac{x_1'+x_2'}{2}$ (as seen in the image plane Fig. 7) and 'N' is the number of

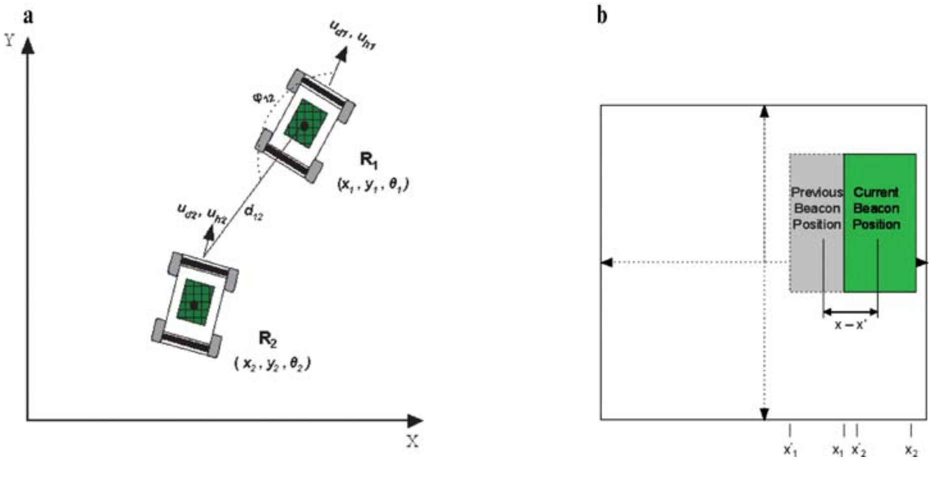

Fig. 7 **a** Leader/follower configuration for two robots **b** Image of beacon in the camera frame

pixels of the visual tag as seen in the camera image plane. $k_{\mathrm{b}1}$, $k_{\mathrm{b}2}$, $k_{\mathrm{d}1}$, and $k_{\mathrm{d}2}$ are the gains to be tuned to estimate the calculated distance and bearing values.

The above leader–follower robot system can be transformed to a new set of kinematic equations where the states of the leader and follower can be taken as input and $\begin{bmatrix} d_{ij} & \varphi_{ij} \end{bmatrix}^T$ as output [13].

$$\begin{aligned} \dot{X}_{12} &= F_1 u_1 + G_1 u_2 \\ \dot{\theta}_{12} &= u_{h1} - u_{h2} \end{aligned} \tag{4}$$

where $X_{12} = \begin{bmatrix} d_{ij} & \varphi_{ij} \end{bmatrix}^T$ is the output of the leader–follower system, $\theta_{12} = \theta_1 - \theta_2$ is the orientation of follower with respect to leader, $u_1 = [u_{d1} \quad u_{h1}]^T$ is the control input to R_1, $u_2 = [u_{d2} \quad u_{h2}]^T$ is the control input to R_2, and

$$G_1 = \begin{bmatrix} \cos\gamma_{12} & \frac{l}{2}\sin\gamma_{12} \\ \frac{-\sin\gamma_{12}}{d_{12}} & \frac{\frac{l}{2}\cos\gamma_{12}}{d_{12}} \end{bmatrix}; \quad F_1 = \begin{bmatrix} -\cos\varphi_{12} & 0 \\ \frac{\sin\varphi_{12}}{d_{12}} & -1 \end{bmatrix} \tag{5}$$

where $\gamma_{12} = \theta_1 - \theta_2 + \varphi_{12}$ and l is the length of the robot.

From Eq. 4, the input u_1 can be independently selected for the leader and the dynamics of the follower robot can be linearized by the control $u_2 = G_1^{-1}(P - F_1 u_1)$. Then, setting $P = \begin{bmatrix} k_1\left(d_{ij}^d - d_{ij}\right) \\ k_2\left(\varphi_{ij}^d - \varphi_{ij}\right) \end{bmatrix}$, and tuning k_1 and k_2 will minimize the separation error $\left(d_{ij}^d - d_{ij}\right)$ and the heading error $\left(\varphi_{ij}^d - \varphi_{ij}\right)$ [13].

4.2 Hardware Platform

The key requirement for the hardware is the ability to implement different circuit combinations to address the needs of different tasks. Upegui and Sanchez [41] exploit the reconfigurability of the FPGAs to load up different circuits at startup and at run-times. Fault recovery based on partial run-time reconfiguration using FPGA was discussed addressed by Will, Marzwell & Chau [45]; Wu & Madsen [47]; Paulsson, Hubner, Jung & Becker [30] among others. In recent times, the design of dynamically reconfigurable robots was address by many researchers. Ferrandi, Marco and Donatella [15] presented a design methodology for dynamic reconfiguration. A reconfigurable mobile robot navigation system was developed by Meng [25]. A framework for modular robotics was discussed in [36, 42]. A detailed description of designing partial and dynamically reconfigurable applications on Virtex-II FPGAs is given in [6, 8, 12, 26, 34, 35, 40, 48]. In the experimental setup described in this paper, reconfiguration of system hardware is accomplished using a Xilinx Virtex-II Pro FPGA based controller card.

The Virtex-II Pro device is a user programmable gate array with embedded PowerPC processor and embedded high-speed serial transceivers [49]. The 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 (Fig. 8). Each CLB also contains logic that implements a four-input look up tables (LUTs) [14]. 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), 16-bit shift registers, or as 16-bit selective RAM memory. Each CLB also has fast interconnect and connects to a generalized routing

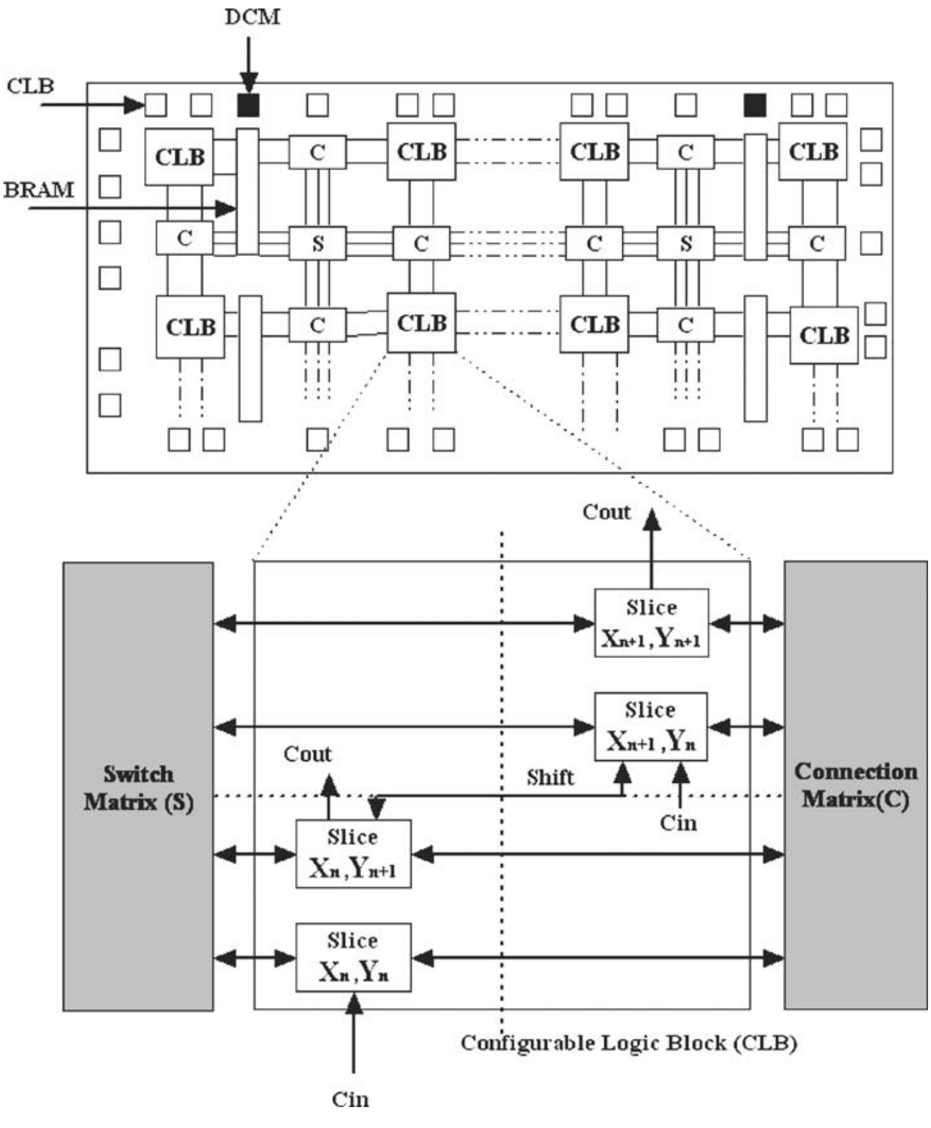

Fig. 8 Internal architecture of Xilinx Virtex II pro FPGA device

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' is used to connect these components. The overall interconnection is hierarchical and is designed to support high speed designs.

The control and management of partial reconfiguration of SRAM-based FPGAs is done through the *JTAG* or *SelectMAP* interfaces. The selection of the interface is based on the *Area Overhead* and the *Reconfiguration Latency*. SelectMAP interface is highly suitable when minimum reconfiguration latency is required. On the other hand, the JTAG interface is preferable if significant control over the placement of reconfigurable modules in the reconfiguration fabric of the FPGA chip is desired [39]. Virtex-II Pro FPGAs also has

Internal Configuration Access Port (ICAP) which provides configuration access for the partial reconfiguration of the FPGA logic. This interface is a subset of the SelectMAP Interface [7, 8]. The ICAP resides in the static portion of the FPGA and care must be taken during reconfiguration to prevent its modification.

The process of implementing a general purpose IO is demonstrated in Fig. 9. The design includes a PowerPC processor core (PPC405) connected to the high bandwidth processor local bus (PLB) and a bridge connecting the PLB and the on-chip peripheral bus (OPB). The required peripherals are simply connected to the OPB. If subsequent reconfiguration of the system requires communication capabilities, say serial communication, then a different module containing a UART can be generated and connected 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.

The Partial Reconfiguration design flow with two partially reconfigurable modules is shown in Fig. 10. Hardware modules not required in the base operating mode are stored as modules in an external memory, while the modules that are common to all the operational modes are created in the static portion of the FPGA. Static Modules (Static Module 1 and Static Module 2 in Fig. 10) communicate with the PR Modules through the *Bus Macros.* On power-up, initial configuration bitstream is loaded onto the FPGA. The initial configuration bitstream consists of the static modules and the initial PR modules. The internal BlockRAMs are initialized with the program code for the PPC405, and the program executes automatically after the configuration is loaded. The program running on the PPC405 then loads the bitstream, and then sends it through ICAP port, partially reconfiguring the PR Module. The new partial bitstream replaces the old partial bitstream. The program running on the PPC405 retains its state throughout the partial reconfiguration process.

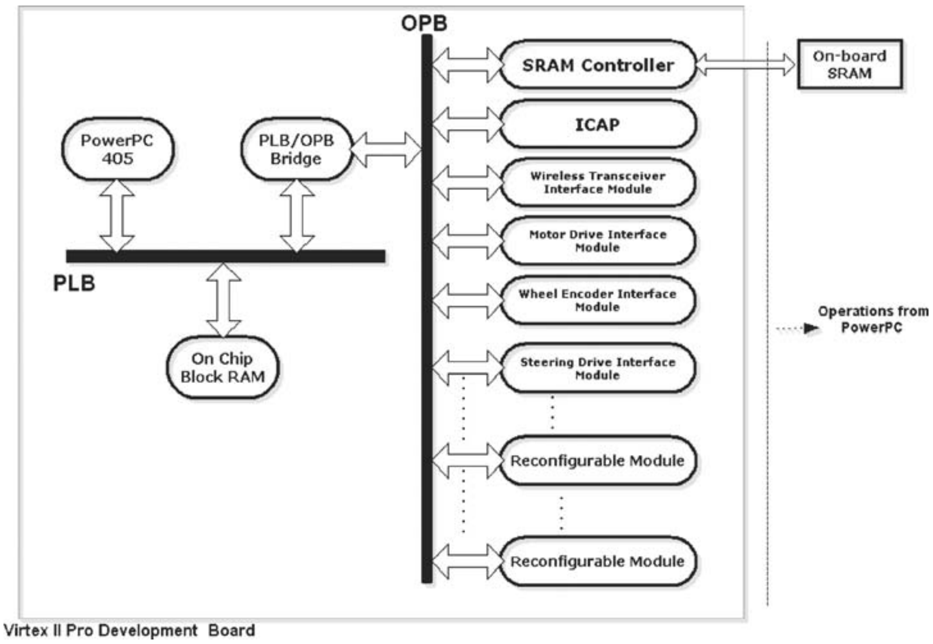
OPB
SRAM Controller
On-board SRAM
ICAP
PowerPC 405
PLB/OPB Bridge
Wireless Transceiver Interface Module
Motor Drive Interface Module
PLB
Wheel Encoder Interface Module
Operations from PowerPC
On Chip Block RAM
Steering Drive Interface Module
Reconfigurable Module
Reconfigurable Module
Virtex II Pro Development Board

Fig. 9 Architecture of the prototype testbed

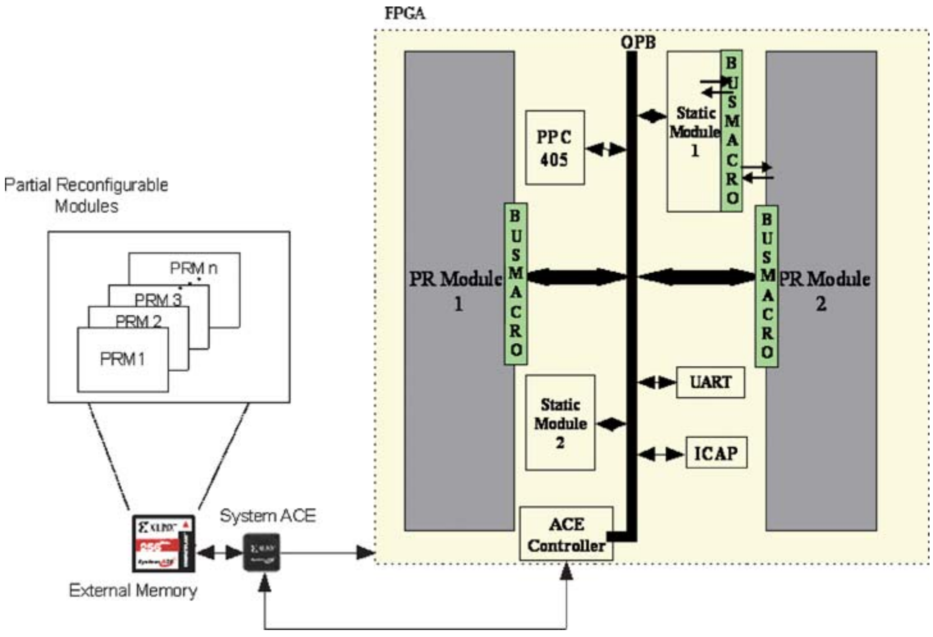

Fig. 10 Partial reconfiguration design flow

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. Partial reconfiguration can be achieved in one of the two ways, namely module-based partial reconfiguration or small-bit manipulations [48]. In the module-based reconfiguration, the entire module is reconfigured. The height of the reconfigurable module is the height of the device and the module can cover one or more columns. In small-bit manipulations, 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.

4.3 Fault Accommodation Using Dynamic Reconfiguration

Dynamic partial reconfiguration in the team of robots can be used to improve system reliability in the event of system faults. Such fault accommodation strategy can be centralized or implemented in an autonomous manner. Sequence diagrams for Centralized and Autonomous fault handling are shown in Figs. 11 and 12, respectively.

In the *Centralized* fault handling, a Control Agent runs on a remote host machine and formulates the strategy to handle the faults occurring in any robot in the team. Given the overall team objectives and the availability of resources, the faulty robot and/or other robots in the team can be reconfigured by the remote agent. On the other hand, in *Autonomous* fault handling, the fault handling strategies are implemented on each of the robots. In the event of a fault, individual robots collaborate with other team members to reconfigure and accomplish a subset of overall task objectives.

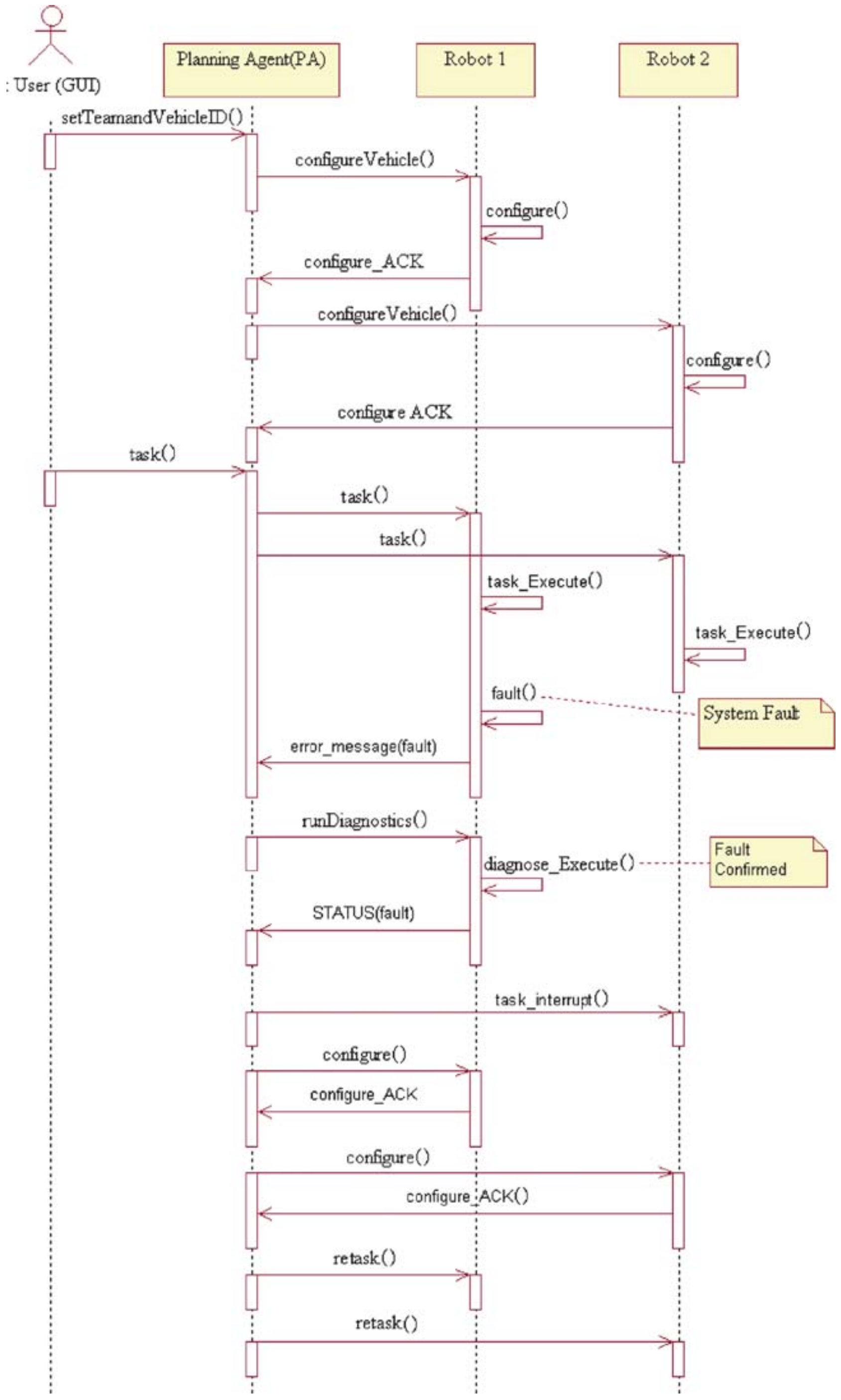

Fig. 11 Sequence diagram showing centralized fault handling and reconfiguration

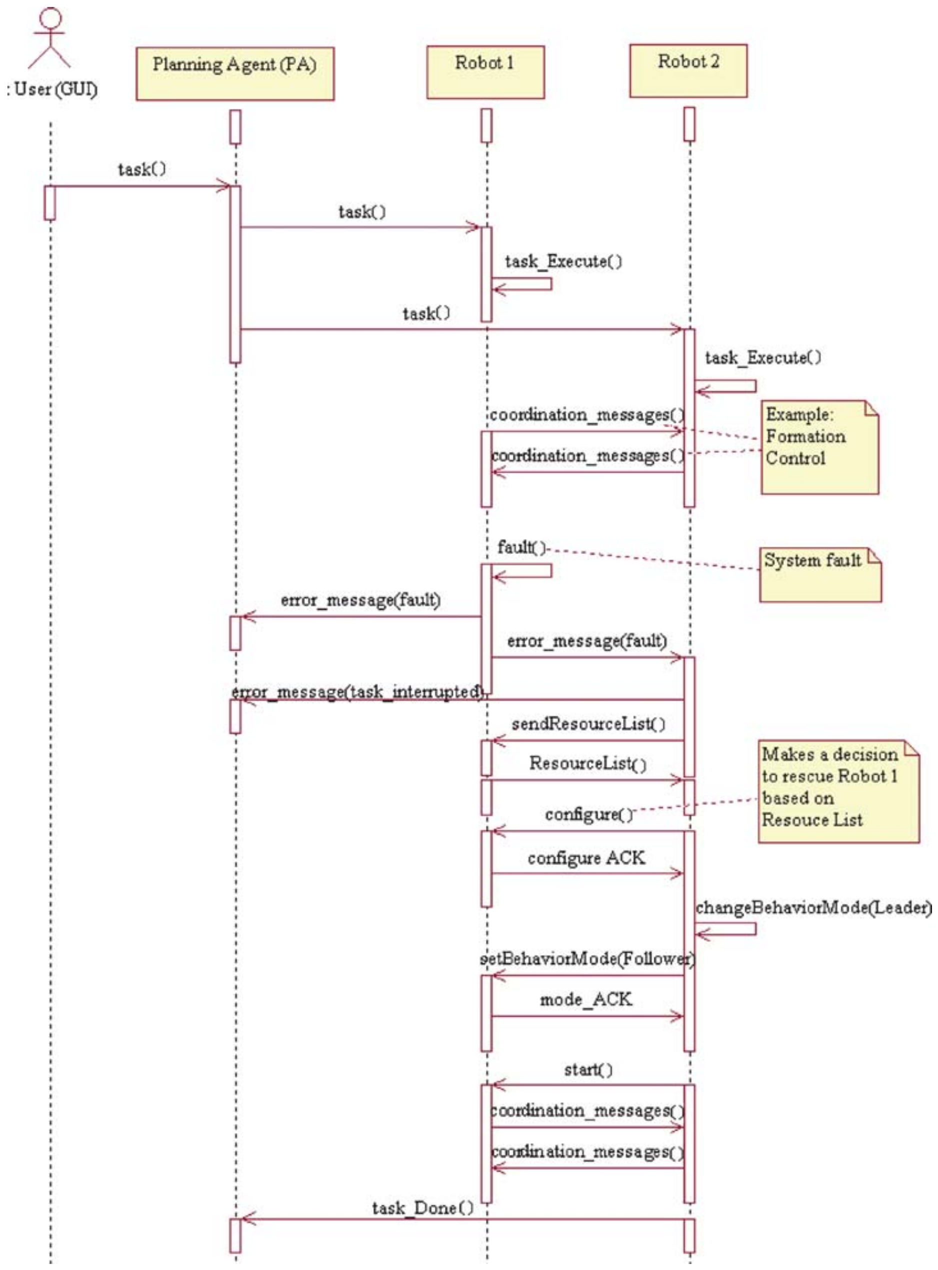

Fig. 12 Sequence diagram showing autonomous fault handling and reconfiguration

5 Experimental Results

The application of reconfigurable computing in the design of autonomous intelligent robots is demonstrated through implementation case studies. In these case studies, the architecture and hardware described in Sections 2 and 3, are implemented on the robotic platform described in Section 4. Using the interfaces specified in Fig. 2, system faults can easily be simulated by overriding the "status" flag of a sensor object [38]. In each control cycle, the sensed values and their status are taken into account and the control strategy is determined. The response to specific faults is determined at design time and can require a combination of software and / or hardware changes. The experimental results in this section demonstrate the dynamic configuration of the robots, modification of runtime behaviors, and reconfiguration based fault handling using both hardware and software modifications.

The prototype test bed of the robot is retrofitted with US Digital EM1 & HEDS-9100-G00 transmissive optical encoders, SHARP GP2D02 Infrared range sensors, a low power multi-channel wireless transceiver (RX-Wi 232 DTS manufactured by Radio-tronix, Inc.), a CMUCam capable of transmitting 17 frames per second, and a HiTec HS-945MG high torque servo for steering. The Mobile base uses the Virtex-II Pro FPGA board (Memec V2PFF672) for on-board processing of vision, control algorithms, and for initiating dynamic reconfiguration of robot behavior. In the base configuration of the robot, only the wheel encoders and the servos are active. This enables the robot to follow set trajectories and accomplish simple navigation tasks. The component diagram for the software implementation is shown in Fig. 13, where the relevant modules are loaded based on the configuration that is chosen. *Reconfiguration of the system results in the simultaneous modification of both the hardware components as well as the associated software modules*. Therefore, only a minimal set of hardware and software components are active at any given instant, thereby improving the efficiency of the system.

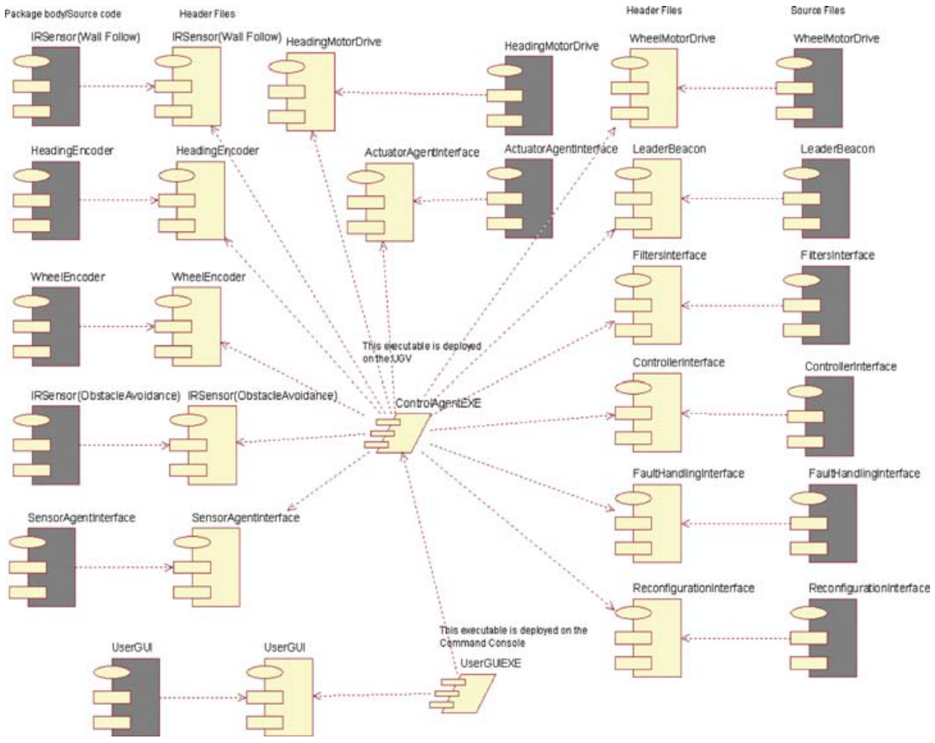

Fig. 13 Component diagram – Software modules and their dependencies

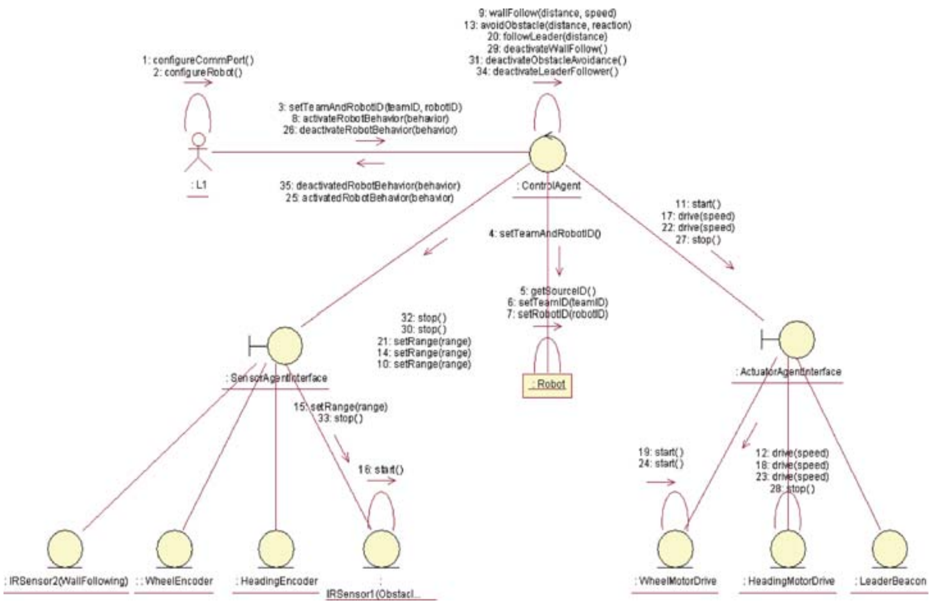

Fig. 14 Collaboration diagram – Activation and deactivation of robot behaviors

a. *Robot Configuration*: At startup, each robot is assigned a unique ID and assigned to a team. The configuration 'spec sheet' on each robot lists the available sensors, their calibration information, and scaling parameters. This information is utilized to download the specific hardware configuration files and base control software on each robot. The collaboration diagram showing the configuration and activation of different performance modes is shown in Fig. 14. In the base mode, navigation is accomplished through the use of the wheel encoders and a PID controller for velocity control. This behavior can be dynamically altered by the inclusion of an obstacle avoidance scheme using infrared proximity sensors or the CMUCam. If obstacle avoidance is implemented

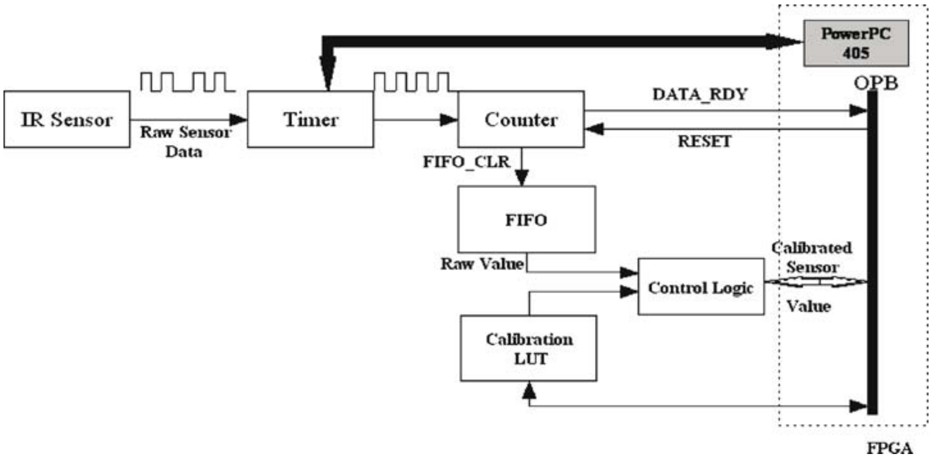

Fig. 15 Processing of raw IR sensor data in hardware

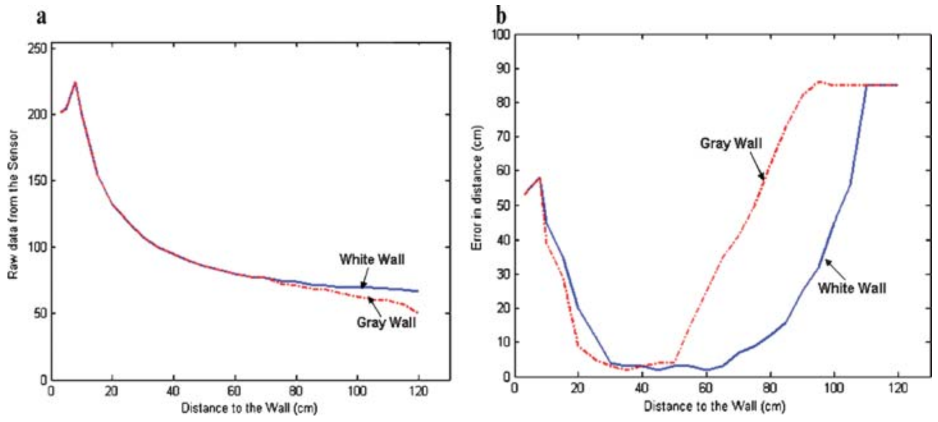

Fig. 16 **a** Distance to the wall vs raw data **b** Distance to the wall vs error

using IR sensors, then the FPGA is reconfigured to route and process the IR signal. The processing of the IR signal in hardware is shown in Fig. 15. The hardware and software combination implements the IR interface and provides the distance to the obstacle in a convenient form to the software. This is in contrast to conventional systems where hardware resources have to be assigned to the sensor irrespective of whether obstacle avoidance is required or not.

b. *Wall Following Configuration*: In order for the robot to navigate along a wall while maintaining the desired separation from the wall, the control algorithm is modified as shown in Fig. 6. Here, two infrared sensors on the side of the robot towards the wall are configured by loading the appropriate VHDL modules into the FPGA. Simultaneously, appropriate software modules are also loaded and the calibration data of the sensors is used to maintain a desired velocity as separation from the wall. The sensitivity of the sensor in detecting a "white" wall versus a "gray" wall is shown in Fig. 16a and b. It can be seen that the output of the sensor is identical for both white as well as gray colored walls up to a distance of 0.8 m. However, the sensor output is nonlinear and linearization of the sensor would result in unacceptable errors for distances below 0.25 m or above 0.4 m. The performance of the sensor can be improved by incorporating a look up table that incorporates the performance data in Fig. 6a. The performance of the robot while navigating along a straight wall and a curved wall is shown in Fig. 17.

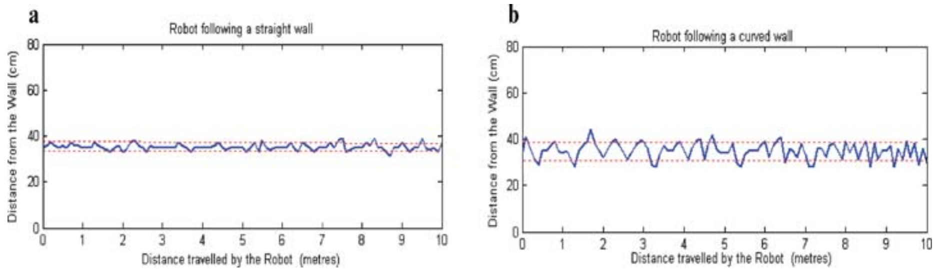

Fig. 17 Wall following behavior: **a** Straight wall and **b** Curved wall

c. *Leader–follower Configuration*: In this example, a lead robot is configured to navigate by following along a wall. The second robot uses the information from the CMU Cam to detect the lead robot and follow it. The CMU Cam is a small, relatively low-cost OV6620 Omnivision CMOS digital camera coupled with an SX28 microcontroller and provides simple high-level image processing data for end-user applications. The CMUCam can identify and track colored objects that fall within given RGB (Red, Green, Blue) values at up to 17 frames per second with 80×143 resolution. It can also provide information about the object such as the center, size (in pixels), and the confidence of detection. In the leader-follower configuration, the lead robot is assigned a unique color tag and the controller for leader–follower configuration is loaded into the second robot. The pixel size of the tag indicates the separation between the leader and the follower. The horizontal distance of the centroid of the tag from the center of the image represents the heading of the follower relative to the leader. Proximity sensors are used to avoid collisions with the wall. The performance of the follower robot is shown in Fig. 18. It can be seen that vision based navigation results in larger errors but frees up the second robot from complex computations for planning and executing specific tasks.

The tracking error in vision based navigation is influenced by the quality of the camera and the sophistication of the image processing software. While the above experiment was designed to illustrate the advantages of hardware/software based reconfiguration, in a practical implementation, a tradeoff between cost and performance must be performed before the selection of camera and other system components.

d. *Autonomous Fault Handling Using Dynamic Reconfiguration of the Robots:* In this example, the ability of the robots to dynamically reconfigure is exploited for fail safe

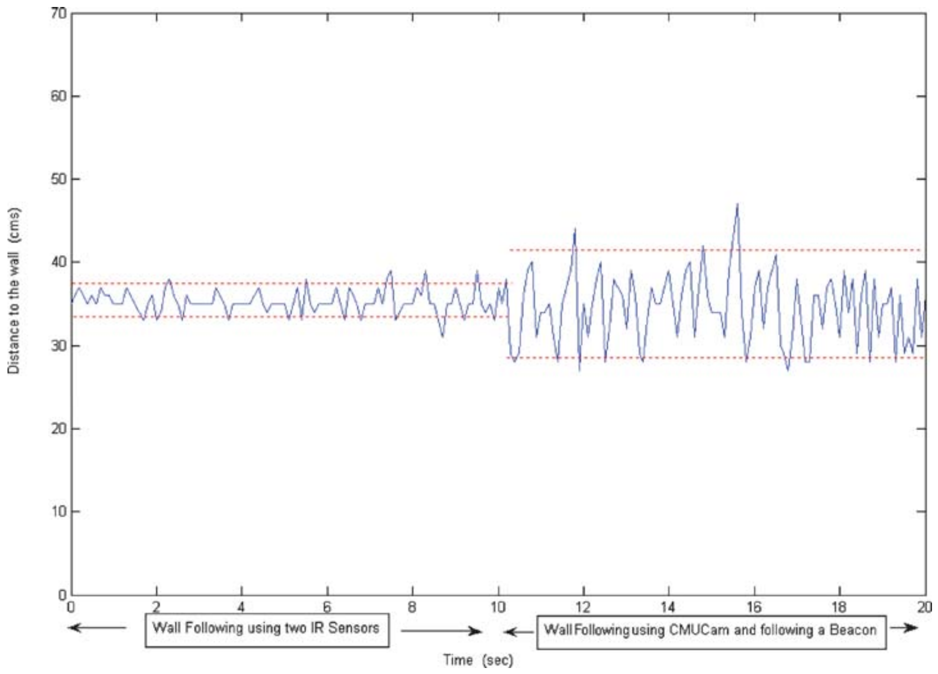

Fig. 18 Vision based tracking using CMUCam

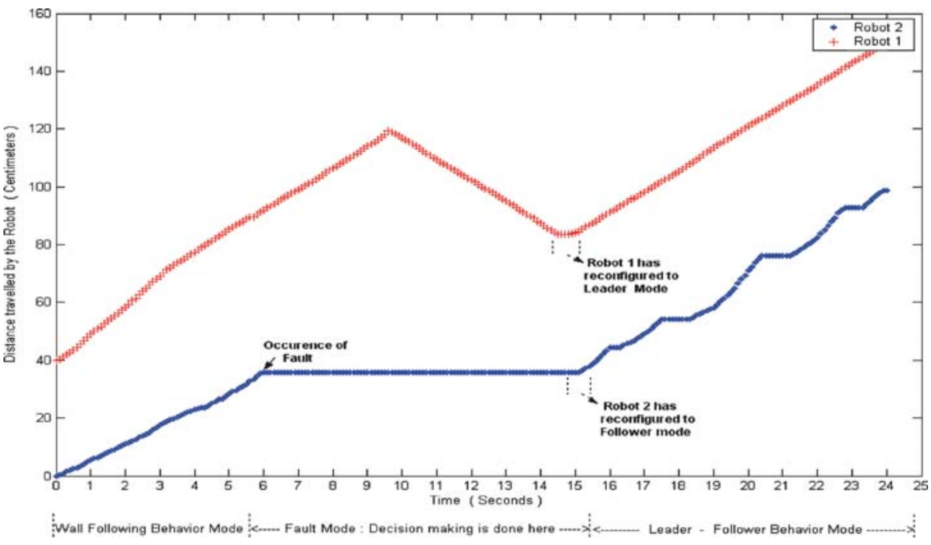

Fig. 19 Dynamic modification of behavior using partial reconfiguration of the hardware

operation and for improved reliability. Here the robot (Robot 2) is executing a task navigating along a wall. The distance from the wall is measured using two infrared sensors mounted on the side of the robot. The location of the robot is updated in each control cycle using the outputs of the two wheel encoders. Sensing failures can be catastrophic and incapacitate a robot. In such an instance, the failure is broadcast as an 'SOS' message and 'Robot 1' in the vicinity responds by approaching the last known location of the failed robot. Robot 1 communicates its presence to Robot 2 and the list of available onboard sensors is exchanged between the robots. The negotiation between Robots 1 and 2 results in the selection of appropriate sensors necessary for both the robots to function as a team in the *leader–follower* mode. In the case study, the onboard

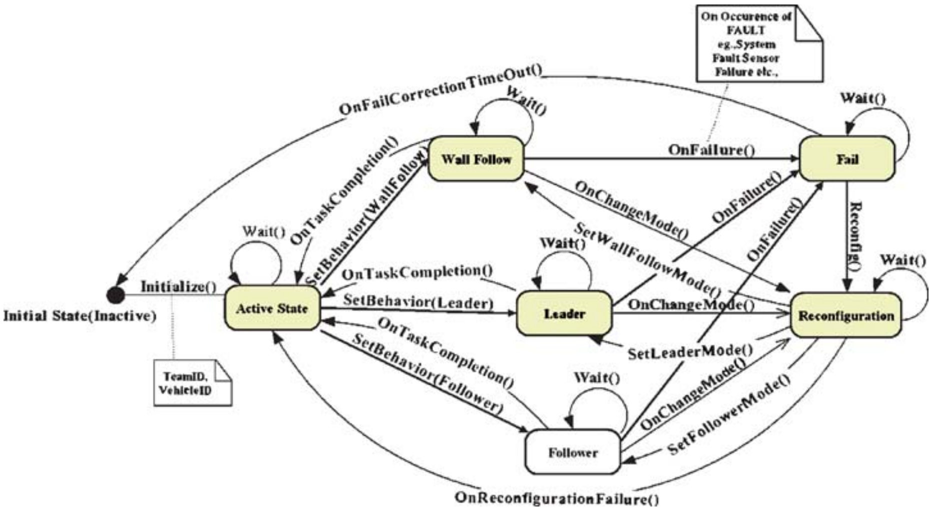

Fig. 20 State diagram of the three-behavior robot

Table 1 Utilization of the FPGA by the Overall design without using partial reconfigurable FPGAs

Module	Occupied slices (total: 4,928)		Equivalent gate count
	Slice count	Percentage	
Static modules only	2,862	58.1%	1,967,380
Static modules + wall following behavior module	2,983	60.5%	2,061,154
Static modules + wall following + leader/follower behavior modules	3,273	66.5%	2,192,918
Static modules + wall following + leader/follower + obstacle avoidance behavior modules	3,463	70.3%	2,380,516
Overall design without using partial reconfigurable FPGAs	3,509	71.2%	2,412,137

camera on Robot 2 is configured and utilized to identify Robot 1. Robot 2 uses the visual data to align itself with respect to the lead robot and then configures itself into leader–follower mode. On reconfiguration, the team functions cooperatively and Robot 1 leads Robot 2 back to its destination. The performance of the robots is shown in Fig. 19. The state diagram depicting the transition between the behaviors of the robots is shown in Fig. 20.

The above example, while simple, illustrates the power of dynamic system reconfiguration to modify functionality, as well as accommodate system failures. While the selection of different behavior modules appears automatic, this can only be accomplished through the creation of appropriate modules and switching criteria at the design time.

6 Summary

In this paper, the development of intelligent robots using reconfigurable computational hardware and software was presented. The proposed design is modular and incurs minimal hardware and computational overhead for each operational mode of the system. Case studies in basic navigation, wall following and leader-following illustrate the design procedure and the benefits of hardware/software co-design.

The basic navigation, wall following, and leader following modules are implemented in the partially reconfigurable modules. It can be seen from Table 1 that the static modules,

Table 2 Utilization of the FPGA by the Overall design when implemented on partially reconfigurable FPGAs

Module	Occupied slices (total: 4,928)		Equivalent gate count
	Slice count	Percentage	
Static modules only	2,862	58.1%	1,967,380
Static modules + wall following behavior module	3,001	61%	2,062,930
Static modules + wall following + leader/follower behavior modules	3,162	64.1%	2,170,551
Static modules + wall following + leader/follower + obstacle avoidance behavior modules	3,188	64.7%	2,190,868
Overall design using partially reconfigurable FPGAs	3,205	65.04%	2,202,382

comprising of the PowerPC core, Bus interface modules, the OPB and PLB buses etc., occupy 58.1% of the available FPGA space (equivalent gate count of 1,967,380) while the ‘wall follower’ behavior module and the ‘leader follower’ behavior modules occupy 2.4% (equivalent gate count 93,774) and 8.4% (equivalent gate count 225,538)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 (equivalent gate count 2,412,137) with the additional software overheads. As the number of behavior modules on the robot increases, the slice count also increases. This means that only a limited functionality can be implemented in static designs. This would a serious problem 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 and the results are shown in Table 2. 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 (PR Module in Figure 21). 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 (shown in Table 2). 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 single robot for performing various tasks without increasing the slice count on the FPGA.

The bit-streams of each of the reconfigurable modules are stored in an external memory (30 MB memory card) and the reconfiguration is accomplished through the SelectMAP interface at 50 MHz clock speed. The average resulting bit-stream for each of the configurations is 1,175 kB resulting in a reconfiguration time of 23.4 ms and a worst net delay of about 9.686 ns. 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. This also leads to an efficient implementation as online learning leads to the refinement of the behavior modules.

7 Conclusions

In this paper, the design of intelligent robots based on reconfigurable FPGA based hardware was addressed. Hardware–Software co-design techniques were used to implement a hierarchical architecture that enables the implementation of fault tolerant designs. Concepts from rapid prototyping were utilized to model and implement the hardware and software components and for the development of teams of intelligent reconfigurable robots. Case studies demonstrate that these robots can be individually configured and tasked at run-time. These systems are also shown to be flexible, robust, and efficient. The case studies also show that this methodology allows for the realization of complex behaviors in robots using FPGA based reconfigurable hardware. The architecture and design methodology presented opens new horizons in the development of evolvable systems that was not possible using traditional design techniques.

References

1. Ahle, E., Soffker, D.: A concept for a cognitive-oriented approach to build autonomous systems. IEEE Conference on Systems, Man, and Cybernetics. **3**(3), 2929–2935 (2005)
2. Antsaklis, P.J., Passino, K.M.: Towards intelligent autonomous control systems: Architecture and fundamental issues. J. Intell. Robot. Syst. **1**, 315–342 (1989)
3. Albus, J.S., Meystel, A.M.: Engineering of Mind: An Introduction to the Science of Intelligent Systems. Wiley Series on Intelligent Systems, New York, NY (2000)
4. Albus, J.S.: Features of Intelligence Required by Unmanned Ground Vehicles. National Institute of Standards and Technology, Gaithersburg, MD (2000)
5. Balch, T., Arkin, R.: Behavior-based formation control for multirobot teams. IEEE Trans. Robot. Autom. **14**(6), 926–939 (1998)
6. Berthelot, F., Nouvel, F.: Partial and Dynamic Reconfiguration of FPGAs: a top down design methodology for an automatic implementation. IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures. **1**(1), 436–437, (2006)
7. Blodget, Brandon, McMillan, Scott, Lysaght, P.: A lightweight approach for embedded reconfiguration of FPGAs. The Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'03). **3**(5), 1530–1531 (2003)
8. Bobda, C., Blodget, B., Huebner, M., Niyonkuru, A., Ahmadinia, A., & Majer, M.: Designing partial and dynamic reconfigurable applications on Xilinx Virtex-II FPGAs using Handel-C. http://www.celoxica.com/techlib/files/CEL-W041223160N-325.pdf (2004)
9. Bonasso, R.P., Firby, R.J., Gat, Erann, Kortemkamp, David, Miller, David. P., Slack, Mark G.: Experiences with an architecture for intelligent, reactive agents. J. Exp. Theor. Artif. Intell. **9**(2), 237–256 (1997)
10. Commuri, S., Sarangapani, J.: Workshop on smart embedded systems for control. IEEE International Symposium on Intelligent Control, Houston, Texas (2003)
11. Commuri, S.: A framework for implementing intelligence in embedded controls. IEEE International Conference on Industrial Electronics and Control Applications (ICIECA 2005) (2005)
12. Danne, Klaus, Bobda, Christopher, Kalte, Heiko: Run-Time exchange of mechatronic controllers using partial hardware reconfiguration. Lect. Notes Comput. Sci. (LNCS) 2778, **9**(4), 272–281 (2003)
13. Das, A.K., Fierro, R., Kumar, V., Ostrowski, J.P., Spletzer, J., Taylor, C.J.: A vision-based formation control framework. IEEE Trans. Robot. Autom. **18**(5), 813–825 (2002)
14. Donti, S., Haggard, R.L.: A survey of dynamically reconfigurable FPGA devices. IEEE Proceedings of the 35th Southeastern Symposium on System Theory. **8**, 422–426 (2003)
15. Ferrandi, Fabrizio, Santambrogio, Marco D., Donatella, Sciuto.: A design methodology for dynamic reconfiguration: the CARONTE architecture. 19th IEEE International Symposium on Parallel and Distributed Processing (2005)
16. Franchino, G., Buttazzo, G., Facchinetti, T.: A distributed architecture for mobile robots coordination. 10th IEEE Conference on Emerging Technologies and Factory Automation. **2**, 149–156 (2005)
17. Galanis, M., Dimitroulakos, G., Goutis, C.E.: Performance improvements from partitioning applications to FPGA hardware in embedded SoCs. J. Supercomput. **35**, 185–199 (2006)
18. Harkin, J., McGinnity, M.T., Maguire, L.P.: Partitioning methodology for dynamically reconfigurable embedded systems. IEEE Proceedings on Computers and Digital Technology, **147**, 391–396 (2000)
19. Hoff, J., Bekey, G.: An architecture for behavior coordination learning. Proceedings of IEEE International Conference on Neural Networks. **5**(1), 2375–2380 (1995)
20. Huntsberger, T., Pirjanian, P., Trebi-Ollennu, A., Nayar, H.D., Aghazarian, H., Ganino, A.J., Garret, M., Joshi, S.S., Schenker, P.S.: CAMPOUT: Control architecture for tightly coupled coordination of multirobot systems for planetary surface exploration. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans. **33**(5), 550–559 (2003)
21. IEEE P1451.1/D2.19. Draft standard for a smart transducer interface for sensors and actuators – Network-Capable application processor (NCAP) information model
22. IEEE 1451.2-1997. IEEE standard for a smart transducer interface for sensors and actuators-transducer to microprocessor communication protocol and transducer electronic data sheet (TEDS) format.
23. Long, M., Gage, A., Murphy, R., Valavanis, K.: Application of the distributed field robot architecture to a simulated demining task. Proceedings of the IEEE International Conference on Robotics and Automation, Barcelona, Spain, pp. 3193–3200 (2005)
24. Maes, P., Brooks, A.R.: Learning to coordinate behaviors. Proceedings of AAAI-90, 796–802. Boston, MA (1990)
25. Meng, Y.: An Agent-based mobile robot system using configurable SOC technique. Proceedings of the 2006 IEEE International Conference on Robotics and Automation, Orlando, Florida, 3368–3373 (2006)

26. Mermound, G.: A module-Based dynamic reconfiguration tutorial. Logic Systems Laboratory. Ecole Polytechnique Federale de Lausanne. http://www.ic2.epfl.ch/~gmermoud/ files/publications/DPRtutorial.pdf (2004)
27. Mesquita, D., Moraes, F., Palma, J., Moller, L., Calazans, N.: Remote and partial reconfiguration of FPGAs: Tools and trends. IEEE Proceedings on International Symposium on Parallel and Distributed Processing (2003)
28. Muthuraman, R., Fajebe, A., Commuri, S.: Intelligence in embedded controls – a case study. IEEE Region 5 Conference on Annual Technical and Leadership Workshop (2004)
29. Parker, L.E.: ALLIANCE: An architecture for fault tolerant multi robot cooperation. IEEE Trans. Robot. Autom. **14**(2), 220–240 (1998)
30. Paulsson, K., Hubner, M., Jung, M., Becker, J.: Methods for run-time failure recognition and recovery in dynamic and partial reconfigurable systems based on Xilinx Virtex-II Pro FPGAs. IEEE Computer Society Annual Symposium on emerging VLSI Technologies and Architectures. **1**, 159–166 (2006)
31. Proctor, F.M., Damazo, B., Yang, C., Frechette, S.: Open architectures for control. Technical Report submitted for National Institute on Standards and Technology NISTIR – 5307 (1993)
32. Sarkar, N., Yun, X., Kumar, V.: Control of a single robot in a decentralized multi-robot system. IEEE Int. Conf. Robot. Autom. **2**, 896–901 (1994)
33. Schreckenghost, D.P., Bonasso, P., Kortenkamp, D., Ryan, D.: Three tier architecture for controlling space life support systems. IEEE Proceedings on International Joint Symposia on Intelligence and Systems, 195–201 (1998)
34. Sedcole, P., Blodget, B., Becker, T., Anderson, J.: Modular partial reconfiguration in Virtex FPGAs. IEE Proc. Comput. Digit. Tech. **153**(3), 157–164 (2006)
35. Lopez-Buedo, Sergio, Grrido, Javier, Boemo, Eduardo I.: Dynamically inserting, operating and eliminating thermal sensors of FPGA based systems. IEEE Trans. Compon. Packag. Technol. **25**(4), 561–566 (2002)
36. Shibamurat, H., Fukuyama, M., Uchida, D., Ikeda, S., Kuga, M., Sueyoshi, T.: EXPRESS-1: a dynamically reconfigurable platform using embedded processor FPGA. IEEE International Conference on Field-Programmable Technology, 209–216 (2004)
37. Simmons, R., Smith, T., Dias, M.B., Goldberg, D., Hershberger, D., Stentz, A., Zlot, R.: A layered architecture for coordination of mobile robots – In Multi-Robot Systems: From Swarms to Intelligent Automata. Proceedings from the 2002 NRL Workshop on Multi-Robot Systems, Kluwer, pp. 103–112 (2002)
38. Commuri, S., Tadigotla, V., Sliger, L.: FPGA-Based design of intelligent robot teams. Proceedings of the IEEE International Symposium on Intelligent Control, Munich, Germany, 1220–1225 (2006)
39. Tan, H., DeMara, R.F., Thakkar, A.J., Ejnioui, A., Sattler, J.D.: Complexity and performance tradeoffs with FPGA partial reconfigurable interfaces. Submitted to the 13th Reconfigurable Architectures Workshop (RAW'06), Greece (2006)
40. Ullmann, Michael, Hubner, Michael, Grimm, Bjorn, Becker, Jurgen.: An FPGA run-time system for dynamic on-demand reconfiguration. IEEE 18th International Parallel and Distributed Processing Symposium (IPDPS '04) (2004)
41. Upegui, A., Sanchez, E.: Evolving hardware by dynamically reconfiguring Xilinx FPGAs. IEEE Conference on Evolvable Systems (ICES'05). Barcelona, Spain (2005)
42. Upegui, A., Moeckel, R., Dittrich, E., Ijspeert, A., Sanchez, E.: An FPGA dynamically reconfigurable framework for modular robotics. 18th International Conference on Architecture of Computing Systems, Innsbruck, Austria (2005)
43. Valavanis, K., Saridis, G.N.: Information-theoretic modeling of intelligent robotic systems. IEEE Trans. Syst. Man Cybern. **18**(6), 852–872 (1988)
44. Tadigotla, V., Commuri, S.: Design and implementation of reconfigurable mobile sensor systems. WSEAS Transactions on Circuits and Systems, **2**(6), 400–408 (2007)
45. Will, Hua Zheng., Marzwell, N.I., Chau, S.N.: In-system partial run-time reconfiguration for fault recovery applications on spacecrafts. Proceedings of IEEE International Conference on Systems, Man and Cybernetics. **4**(4), 3952–3957 (2005)
46. Wills, L., Kannan, S., Sander, S., Guler, M., Heck, B., Prasad, J.V.R., Schrage, D., Vachtsevanos, G.: An open platform for reconfigurable control. Control Syst. Mag. **21**(3), 49–64 (2001)
47. Wu, K., & Madsen, J.: Run-time dynamic reconfiguration: a reality check based on FPGA architectures from Xilinx. IEEE NORCHIP Conference (2005)
48. Xilinx, Inc.: Two flows for partial reconfiguration: Module based or Difference based. Xilinx Application Note XAPP290, http://www.xilinx.com/bvdocs/appnotes/xapp290.pdf (2004)
49. Xilinx, Inc.: Virtex-II Pro Platform FPGA User Guide, version 1.8. http://www.xilinx.com/bvdocs/userguides/ug012.pdf (2005)