

An Agent-based Reconfigurable System-on-Chip Architecture for Real-time Systems

Yan Meng

Stevens Institute of Technology, Hoboken, NJ 07030, USA

ymengl@stevens.edu

Abstract

For a real-time system who must adapt to the highly dynamic environments, instead of using a general processor, a reconfigurable multi-core system-on-chip (RMCSoc) architecture is applied in this paper. A unified agent-based hardware/software co-design methodology is proposed to optimize the system performance on this RMCSoc, and a self-reconfigurable platform is proposed to enable the FPGA to dynamic reconfigure the hardware at runtime by an embedded processor. Due to the multi-agent implementation, a new communication mechanism between the agents is also presented. This proposed embedded framework is suitable for most real-time applications, such as mobile robots, embedded video systems, and even distributed real-time systems with enhanced communication capability.

1. Introduction

Conventionally, all the tasks of a complex real-time system are implemented on a general processor in software only due to its great flexibility. A powerful processor is needed to guarantee that hard deadlines always are met, even for very rare conjunctions of events. Sometimes the sensors of a real time system are supposed to work in parallel and independently, it would be difficult to implement the parallel sensing in software. Furthermore, with multi-tasks, signals, events, it would be a challenging job for a software engineer to implement all of the tasks under real-time constraints especially with some computation-intensive information, say video images.

On the other hand, the specific hardware can respond to external inputs more efficiently since the multiple hardware units can all operate in parallel, concurrent, and asynchronous, so that individual response times are much less variable and easier to guarantee, even as the number of tasks increases. Parallel hardware is not so affected by issues such as task swapping, scheduling, interrupt service, and critical sections, which complicate real-time software

solutions. The expensive ASIC can fulfill the speed criteria. However, it is a complicated and expensive procedure if a slight change occurs. The newest FPGA technology allows a designer to use a single reconfigurable platform to instantiate both the processors and the required logic units, which directly leads to feasibility of the reconfigurable multi-core system-on-chip (RMCSoc) architecture.

This RMCSoc architecture raises the possibility of different co-design approaches. The basic hardware/software co-design flow starts with partitioning a description of the application into hardware and software components. Then on each component, the processes are scheduled based on their priorities or deadlines. Due to the different behaviors of the two environments, co-simulation techniques are necessary to approximate the environments and their interface. If the co-simulation result does not meet the specifications, the designer may have to re-partition the design. Otherwise, the design can be co-synthesized and implemented on the target platform.

To speed up the system integration and hardware/software co-design process on a RMCSoc platform, a unified hardware/software interface is necessary. This paper presents an agent-based co-design methodology running on a RMCSoc platform, where multiple soft-core processors instantiated on an FPGA. This unified methodology is developed to design high-level aspects of systems. The system is first decomposed into several agents based on system specifications, where these agents can accomplish some specific tasks independently and can communicate with each other. These agents are then partitioned into software agents and hardware agents. Since all of the agents share the same mechanism, on top of this unified agent-based representation, it is possible to provide a unified communication mechanism between the hardware and software parts. A novel on-demand message passing (ODMP) communication protocol for this multi-agent system is then proposed. This communication mechanism provides transport independence and therefore, it is portable to different agent-based real-time systems without significant modification to the both software

and hardware. The only parts need to be customized are adding different transport-dependant adapters to new systems.

This RMCSoc platform is also desirable for those real-time systems which must be highly responsive to the dynamic environments. Since hardware component is more efficient to handle external events compared to software component, runtime switching some sensors for a robot system, which is necessary to maintain the system performance, can be effectively managed by a dynamic reconfigurable hardware logic units. In this paper, a new self-reconfiguration platform is proposed. Self-reconfiguration is a special case of dynamic reconfiguration where the configuration control is hosted within the logic array that is being dynamically reconfigured. The part of the logic array containing the configuration control remains unmodified through out execution. There are several advantages of such a design. Firstly, the control logic is located as close to the logic array as possible, thus minimizing the latencies associated with accessing the configuration port. Secondly, it provides a simple mechanism to implement our previously proposed multi-agent communication protocol, thus fewer separate discrete devices are required, reducing the overall system complexity.

The remaining sections of the paper are structured as follows. Section 2 describes the previous work done on hardware/software co-design on reconfigurable embedded platforms as well as the hardware self-reconfiguration. Section 3 presents the multi-agent system architecture. A self-reconfiguration platform is presented in Section 4. Then agent-based hardware/software co-design on a RMCSoc platform is introduced in Section 5. Some preliminary experimental results are given in Section 6. Finally, Section 7 summarizes the conclusion and future work.

2. Related Work

Most of the previous work on reconfigurable embedded platform for real-time systems is implemented on software only. Gafni *et al* [7] proposed an architectural style for real-time systems, in which the dynamic reconfiguration is implemented by a synchronous control task in response to the sensed conditions. To handle run-time dependencies between components, [2] employ run-time analysis of interactions to selectively determine which interactions should be allowed to proceed, in order to reach a safe state for change, and which to block until reconfiguration completed. Cobleigh *et al* [4] presented a hierarchically self-adaptation models for

the robot system to provide fault-tolerance. MacDonald *et al* [8] proposes some design options for dynamic reconfiguration with their service-oriented software framework. Stewart *et al* [11] proposes a programming paradigm that using domain-specific elemental units to provide specific guidelines to control engineers for creating and integrating software components by using port-based object. [13] [10] present a self-reconfigurable robot design with a distributed software architecture that follows the physical reconfiguration of hardware in response to the needs of task or environment.

The recent advent of dynamic reconfigurable FPGA platform attracts more attentions in large applications. Most of the real-time systems use FPGAs to speed up the portions of an application that fail to meet required real-time constraints. The approach of applying reconfigurable logic for data processing has been demonstrated in some areas such as video transmission, image-recognition and various pattern-matching operations [9, 14]. In [16], the system uses FPGAs that are dynamically reconfigured on Dynamically Programmable Gate Arrays at runtime to implement different functions.

To fully take advantage of the reconfigurable characteristics of FPGAs, some self-reconfiguration mechanisms are proposed. Blodget *et. al* [3] proposed a self-reconfiguration platform for embedded system using ICAP and an embedded processor core within the FPGA. Fong *et. al.* [6] developed a system that uses the RS-232 port to transfer configuration bitstreams to the FPGA, which easily leads to a bottleneck for the transfer speed and an increase the reconfiguration latency.

As the soft processor cores for FPGAs become available in the RMCSoc platform, the hardware/software co-design issues becomes critical in the overall system design procedure. Some of the more recent hardware/software co-design research uses reconfigurable processors as the platform for implementation [14, 15], where their architectures combine a reconfigurable functional unit with the microprocessor. Partitioning is a well-known NP-complete problem and many heuristics have been proposed to guide the partitioning of a system into hardware and software components, for example, [17, 18]. In [18], the hardware/software partitioning problem is modeled as a Constraint Satisfaction Problem (CSP), and a genetic algorithm based approach is proposed to solve the CSP to obtain the partitioning solution.

3. Multi-Agent System Architecture

3.1. BDI Agent Model

An *agent* is an independent processing entity that interacts with the external environment and the other agents to pursue its particular set of goals. The agent pursues its given goals adopting the appropriate plans, or intention, according to its current beliefs about the state of the world, so as to perform the role it has been given. Such an intelligent agent is generally referred to as a Belief-Desire-Intention (BDI) agent. In short, belief represents the agent's knowledge, desire represents the agent's goal, and intention lends deliberation to the agent. The agent control loop is: first determine current beliefs, goals and intentions, then find applicable plans and decide which plan to apply, and finally start executing the plan.

The architecture of the BDI agent model is shown in Figure 1, which has three external ports: inter-agent communication, control, and input/output. The control port has three functions: (a) for the system to activate/deactivate the agents; (b) for the agent to inform the system when it finishes its job; (c) for the agent to synchronize with the system. The input/output port is used to send and receive information to and from the host environment. The inter-agent communication port allows the agents to send/receive information with other agents and cooperate with others.

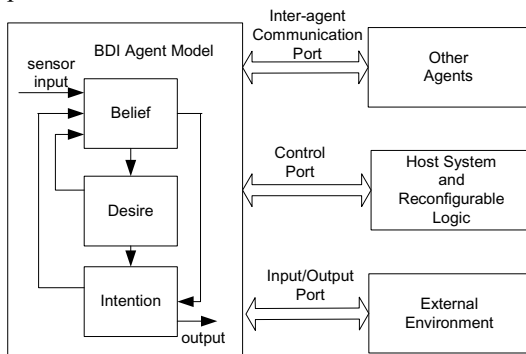


Figure 1: The architecture of BDI agent model. Individual agents are assumed to have intentions and commitments both with respect to goals and plans. Beliefs and desires influence each other reciprocally, and beliefs and desires both influence intentions.

Under the BDI model, agents may be given “pre-compiled” agents, or they may plan or learn new plans at execution time. Giving BDI agent pre-compiled plans is a method for ensuring predictable agent under critical operational conditions, and for ensuring

performance. BDI agents are highly suitable for the development of time and mission critical systems, as the BDI approach provides for the verification and validation of the model.

3.2. Multi-agent Communication Mechanism

Agents are generally designed with a specific purpose in mind. They do one or perhaps several tasks very well, but aren't often designed as a jack-of-all-trades. If agents must perform more tasks, we can either increase their complexity (which increase the development effort), or we can make them work cooperatively. Usually a complex real-time system can be constructed as a set of independent and cooperating agents, where each agent owns its own intention and pursuit its own sets of goals. For cooperation between agents to succeed, effective communication is required. We can view a collection of agents that work together cooperatively for a global goal. For a global goal to function coherently, we need a common language and communication medium.

Various agent communication languages have been developed, such as Knowledge Query Manipulation Language (KQML) [1], and Foundation for Intelligent Physical Agent (FIPA) [5]. But they have high overheads in terms of memory usage and computation time that are not always acceptable for real-time systems. Numerous proprietary methods also have been developed, but more and more often they encounter maintenance, porting, and enhancement issues.

Since all of the agents work in an asynchronous mode, the messages will only be issued on demand. This leads to a new developed on-demand message passing (ODMP) protocol in this paper. A typical example of two agents connected by a communication path is shown in Figure 2. In this mechanism, each agent creates a message queues queued in FIFO order with priorities, where the messages marked as urgent are attached to the head of the queue, and the ones marked as normal are lined in a FIFO order.

Each agent may have multiple current tasks need to be implemented. The database consists of group database, which maintains the list of all other agents in the system along with their current status, and agent dependency database, which maintains the list of other agents that are required for the current agent to process along with the data from last communication. For example, before a task in agent 1 sends a message to agent 2, agent 1 needs to check the status of agent 2 in its database. If the agent 2 is not on service any more due to some failures, agent 1 stops all its messages to agent 2. On the other hand, when agent 1 receives a

message from agent 2, it makes its decision based on the new message and updates its own database as well.

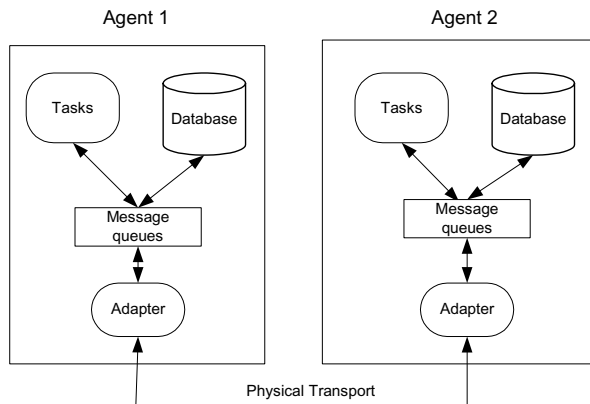


Figure 2: Multi-agent communication structure

In order to provide a media-independent agent communication mechanism, adapters are necessary. Adapters provide a uniform interface to agent, regardless of the particular transport used. The transport can be a bus driver for software agent, or other high-level communication protocols, such as UDP. When a message is sent to an agent (either software or hardware), the agent passes the message buffer to the adapter. The adapter sends data to the agent by way of the supported transport. Similarly, when a message is received from an agent, an adapter reconstructs a message buffer from the incoming data and sends buffer to the agent.

The minimum message transmitting/receiving unit is called message frame. A message may contain multiple message frames. The maximum number of message frames for a single message is platform dependent, which is restricted by the available internal memory and external memory. A message frame includes protocol header, service header, and data to be transmitted. The protocol header is defined by the physical transport, which was built from values provided by the adapter. The contents of this header may vary from protocol to protocol, but should include fields such as source ID, destination ID, sender ID, receiver ID, data size, and priority of messages if the transport supports priority. The sender and receiver are the broker agents the messages have to pass through to the destination. Service header usually identifies the internal service sending the message and the message type, which are application dependant.

When sending a message to a remote message queue, this data can be either the entire message or a fragment of the message to be sent if the total message size exceeds a message frame. A message buffer is needed to reconstruct the whole message if it has been

segmented before the transmission. The agent does not acknowledge individual message unit, instead, only the whole messages are acknowledged. If the event of a transmission error or if a telegram is lost, the whole message must be re-transmitted.

4. A Self-Configuration Platform

For some real-time systems under highly dynamic environments, it is desirable for the sensor agents to be reconfigured at runtime to adapt to the unexpected events. Dynamic reconfiguration implies that an active array may be partially reconfigured, while ensuring the correct operation of those active circuits that are not being changed. Since the Vertex Pro II FPGA is chosen as the RMCSoc platform in our system, the internal reconfiguration access port (ICAP) in Vertex Pro II FPGA makes the self-reconfiguration possible. Some self-reconfiguration platforms proposed in [3] [6] used a limited amount of on-chip memory to store the reconfiguration data, which may easily lead to a performance bottleneck for a complex real-time system, such as a mobile robot.

To overcome this performance bottleneck, a new self-reconfiguration framework using DDR external memory is proposed here. The goal of this framework is to provide a flexible method for implementing field self-reconfiguration while minimizing both system hardware and required configuration data, and reducing the reconfiguration time. As shown in Figure 3, the fixed logic includes common fixed logic circuits, ICAP, configuration controller, multiple dual-port block RAMs (BRAMs), embedded processors, and external memory. The embedded processor, which is one of the soft core PowerPC405s on FPGA, provides intelligent control of the device reconfiguration at runtime. The embedded processor communicates with FPGA logic through IBM-designed CoreConnect bus architecture. The DDR external memory is connected to the processor through process local bus (PLB bus).

The fixed logic part is located the right-most columns because ICAP in Vertex II Pro FPGA is located at the lower-right corner while the reconfigurable logic part is located on the left-side columns. The configuration controller uses the ICAP to reconfigure the reconfigurable logic parts. The dual-port characteristics allow the BRAM to be accessed from different clock domains on each side. One port is accessed from the configuration controller, while the other port is accessed from the reconfiguration logic part.

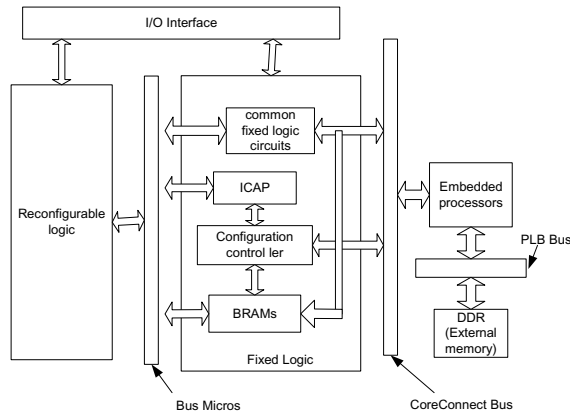


Figure 3: A self-reconfiguration platform

To improve the reconfiguration performance, we can take advantage of the BRAMs in two ways. First, they can be set up as registers with individual address and easily to read from and write to by both configuration controller and reconfiguration logic part. This feature would allow the reconfiguration logic part get setup values from embedded processor via the configuration controller, and also report back the current status to the processor, acting like a shared memory between the processor and reconfiguration logic parts. Second, the BRAMs can be applied as buffers between the configuration controller and reconfigurable part. This buffer feature can reduce the reconfiguration latency due to its pipeline mechanism.

The common fixed logic circuits can implement some specific hardware agents as well as interrupt generation logic. The BRAMs can also be accessed by common fixed logic directly. These BRAMs should be different memory locations than those accessed by configurable logic parts. This feature provides the shared memory mechanism between the common fixed logic circuits and the embedded processor.

Due to the dynamic environment in real-time systems, the reconfigurable logic parts need to be reconfigured in response to the new environment. Usually, there are two trigger conditions for the reconfiguration, one is from bottom-up, and the other is from top-down. The bottom-up method is through event interrupts invoked by the sensor agents if the sensor agents set up some thresholds on the input sensing data. The top-down method is that the processor makes the decision due to the input information from sensor management agent, for example, the incoming perception has dramatic variation comparing to the previous information or the data does not make sense at all. To improve the fault tolerance of the real-time system, both mechanisms are

applied to trigger the reconfiguration process in our platform.

Another benefit from this self-reconfiguration platform is that the multi-agent communication mechanism can easily take advantage of the shared memory feature provided in this platform. Due to the dual-port characteristics of the BRAM, the software agents can send messages to BRAM through configuration controller, while the hardware agents can access their own message queues directly from the same BRAM. On the other hand, if a hardware agent sends a message to a software agent either through the configuration controller if it is a configurable hardware agent, or directly if it is fixed hardware agent, via CoreConnect Bus, the message would be stored on the DDR SDRAM via CoreConnect PLB bus.

In summary, this self-reconfiguration platform provides two advantages: firstly, it provides an autonomous method for the hardware agents to be reconfigured due to dynamic environment for mobile robot system under real-time constraints. Secondly, it provides a feasible physical mechanism to allow the ODMF communication protocol to be implemented on this platform using shared memory features.

5. Hardware/Software Co-Design

Hardware/software co-design provides a way of customizing the hardware and software architectures to complement one another in ways which improve system functionality, performance, reliability, survivability, and cost/effectiveness. There are several issues need to be addressed for the co-design in this section: hardware/software partitioning, and implementation of agent-based architecture in hardware and software.

Automatically partitioning is a challenge problem not only because it is a well-known NP-complete problem, but also due to the lack of efficient and accurate performance profiling tools. To simplify the problem, we propose a heuristic partitioning method here. In general, more regularly structured agents that have highly repetitive and extensive time consuming operations are suitable for implementation in reconfigurable hardware, whereas the more complex and irregularly structured agents should be programmed in software. Another general partitioning policy is that the agents with hard deadlines are distributed to hardware, while the agents with soft deadlines may be implemented in software. We leave the real-time partitioning algorithms to our future research.

Second issue is how to implement the agent-based architecture in software and hardware design. There are some agent-oriented implementations developed, such as JACK [1] and distributed multi-agent reasoning system (dMARS) [5]. In our systems, the software agent follows the similar idea of JACK. The BDI agents with beliefs, desires, intentions, and plan library are created. Events alert the agent to changes in the world or in its internal state. Reactive and proactive events are defined differently and result in a different behavior when executing plans. When an event is raised, the agent looks through its plan library and finds a plan that is relevant to the event and its current beliefs. It then creates an instance of this plan and tries to execute it, however if it fails, another plan is tried until all relevant plans are exhausted.

The hardware agent implementation is similar to the software agent except that in software, all the parameters of the agents can be transmitted by function calls in C/C++, while in hardware we have to specifically define all of the hardware agent entities, their associate ports and parameters in VHDL.

6. Experimental Results

6.1. A Mobile Robot Platform

In order to evaluate the performance of our proposed agent-based methodology running on RMCSoc platform, a Pioneer 3DX mobile robot system developed by ActivMedia Robotics is adopted as a real-time system in our experiment. The rugged P3-DX is 44cm x 38cm x 22cm aluminum body with 16.5cm drive wheels. The two motors use 38.3:1 gear ratios and contain 500-tick encoders.

The on-board sensors of this robot system includes one CCD camera, one night vision gear (NVG) for vision perception at night or dark environments, one laser rangefinder for range measurement, one bumper sensor, a ring of eight sonars, and two motor encoders for odometry measurement.

The multi-agent architecture of a mobile robot system is depicted in Figure 4. The sensor agents, the sensor fusion agent, the motion manager agent, the motion agents, and emergency stop agent are implemented in hardware, while the path planning agent, obstacle avoidance agents, and robot/human interface agent are implemented in software. If each sensor is designed as one sensor agent, then totally number of agents for our mobile robot system is 22 agents, which including 14 sensor agents, 2 motion agents, plus 6 other agents shown in Figure 4.

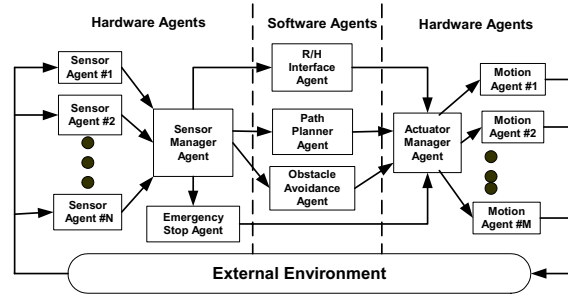


Figure 4: Multi-agent architecture for a mobile robot navigation system

6.2. A RMPSoc Prototype Platform

The Xilinx ML310 embedded development platform is installed to the robot as an on-board processor and our prototype platform. The ML310 offers designers a Virtex-II Pro XC2VP30-based embedded platform. It provides 30k logic cells, over 2,400kb BRAM, and dual PPC405 processors, as well as onboard Ethernet MAC/PHY, 256M DDR memory, multiple PCI slots, and standard PC I/O ports.

By using ML310 embedded platform, hardware agents are configured as fixed or reconfigurable FPGA logic circuits in VHDL, where bus macros are used as fixed data paths for signals going between reconfigurable logic parts and other logic parts (fixed or reconfigurable). Software agents are implemented on the embedded soft cores PowerPC 405 in C/C++.

The IBM-developed hierarchical CoreConnect bus architecture is used to connect the external memory, I/Os, and other peripherals. Although a unified design representation is applied to hardware and software agents, the physical communication mechanism between these two has to go through the CoreConnect bus. Based on the proposed ODMF communication mechanism, the message queues are created individually for each agent (hardware or software). The message queue is saved on internal BRAM for hardware agents and external DDR SDRAM for software agents. If a bus architecture, like CoreConnect bus, consists of three different buses, the OPB, PLB, and DCR, there will be three different adaptor each targeting an individual bus type. For reconfigurable hardware agent, another adaptor is needed for the bus micros.

For a mobile robot system with 22 agents, 32 bits may need for the communication protocol header, where 5 bits for source agent IDs, 5 bits for destination agent IDs, 5 bits for sender agent IDs, and 5 bits for receiver agent IDs, 8 bits for data size, and 4 bits for message priority. 4 bits are needed for the service

header, which include message types, such as stop command from R/H interface agent to motion manager agent, and new event from one sensor agent to sensor fusion agent. Since only the extracted sensor data instead of the raw sensor data will be sent to sensor fusion agent, up to 256 bits are needed for message data.

6.3. Experimental Results

To evaluate the performance of the proposed agent-based approach, the real world experimental scenarios are shown in Figure 5. The mobile robot (red circle) needs to navigate itself from one fixed starting point at room B with good illumination to the fixed destination point at room A with dark illumination while searching a specific feature target (green cube) on its way. The map of two office rooms is given to the robot, where the obstacles (blue) randomly scattered at both rooms are not provided in the map.

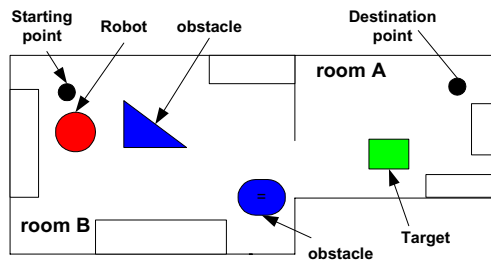


Figure 5: The navigation environment

By implementing both the hw/sw co-design and software only approaches to the mobile robot, the robot has successfully navigate itself to the destination point without hitting any obstacles on its way, and it also detected the target. When the robot is at room B, CCD camera is active while the NVG is inactive, while at room A, it is the other way around. One underlying assumption for the robot navigation system is that if the robot can not figure out its current location or feel confused about its current sensing information, it would stop until it can recover its current location or feel confident about its sensing data. The experiment demonstrates that the proposed agent-based reconfigurable architecture is feasible and can work efficiently for a complex real-time system under highly dynamic environments.

To further evaluate the performance of reconfigurable hardware, several experiments have been conducted with different number of obstacles. The overall runtime for software-agent-only and hw/sw co-design approach are listed in Table 1.

Table 1: Navigation Time Comparison

# of obstacles	Navigation time (minutes)	
	HW/SW	SW only
1	8	12
2	9	15
3	10	17
4	11	19

It can be seen from Table 1 that there is a dramatic decrease in navigation time in a dynamic environment with HW/SW co-design comparing with SW-only approach. The hw/sw codesign approach increases the speed of action and reaction to dynamic environment because it makes decisions much faster than the software competitors. Another reason for the speedup is because hardware responds favorably to dynamic changes during runtime due to its parallel structures.

In the future, the internal execution time for both the approaches will be calculated. It is expected that the speed of hw/sw co-design approach should be over an order of magnitude greater than their equivalent software-based approach.

7. Conclusions

This paper proposes a self-reconfigurable agent-based embedded platform for real-time systems. The major contributions of this paper includes: (a) propose an innovative multi-agent based architecture framework by which a unified hardware/software co-design methodology is applied to improve the system performance and simplify the system design; (b) present a unified transport-independent communication mechanism for multi-agent systems, where an on-demand message passing (ODMP) communication protocol is proposed; (c) establish a self-reconfiguration platform where the configuration control logic is located inside the FPGA logic array and is controlled by embedded processor core. (d) A real-time system example, a mobile robot, has been applied to evaluate the proposed embedded platform for its feasibility and efficiency.

Since the quality of the hardware/software co-design is dependent on the partitioner's allocation of resources, the simple partitioning rules applied in this paper may not fully express the advantages of the proposed agent-based architecture. With the more advanced partitioning algorithm, which is our future research, the proposed agent-based architecture would get better performance.

8. References

- [1] Agent-Oriented Software Pty Ltd, JACK Manual (v4.1), www.agent-oriented.com, 2003.
- [2] J. Almeida, M. Wegdam, M. Sinderen, and L. Nieuwenhuis. Transparent Dynamic Reconfigurable for CORBA. *Proceeding of the 3rd International Symposium on Distributed Objects & Applications (DOA 2001)*, Sept. 17-20, 2001, Rome, Italy.
- [3] B. Blodget, P. James-Roxby, E. Keller, S. McMillan, P. Sundararajan. A self-reconfiguring platform. *Proceeding of the 13th International Conference on Field Programmable Logic and Applications (FPL'03)*, pp. 565-574, 2003.
- [4] J. Cobleigh et al. Containment units: a hierarchically composable architecture for adaptive systems. *ACM SIGSOFT Software Engineering Notes*, 27(6):159-165, November 2002.
- [5] M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge. A formal specification of dMARS. *Proceedings of the 4th International Workshop on Agent Theories, Architecture, and Language*, vol. 1365 of LNAI, pp. 155-176, Berlin, 1998.
- [6] R. Fong, S. Harper, P. Athanas. A versatile framework for FPGA field updates: an application of partial self-reconfiguration. *Proceedings of the 14th IEEE International Workshop on Rapid Systems Prototyping (RSP'03)*, 2003.
- [7] V. Gafni. Robots: a real-time systems architectural style. In *Proc 7th European software engineering conference*, pages 57-74, Toulouse, 1999.
- [8] B. MacDonald, B. Hsieh, and I. Warren. Design for Dynamic Reconfiguration for Robot Software. *2nd International Conference on Autonomous Robots and Agents*, December 13-15, 2004, Palmerston North, New Zealand.
- [9] Y. Meng. A Dynamic Self-reconfiguration Mobile Robot Navigation System. *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2005)*, 2005.
- [10] S. Patterson, K. Knowles, and B. E. Bishop. Toward Magnetically-Coupled Reconfigurable Modular Robots. *Processing of IEEE International Conference on Robotics and Automation*, New Orleans, LA, April 2004.
- [11] D. Stewart, R. Volpe, and P. Khosla. Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects. *IEEE Trans. On Software Engineering*, vol. 23, no. 12, December 1997.
- [12] J. Villascnor and W.H. Mangionc-Smith. Configurable computing. *Scientific American*, no. 6, 1997.
- [13] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, and S. Kokaji. Self-reconfiguration Modular Robots – Hardware and Software Development in Aist. In *Proceedings, IEEE International Conference on Robotics, Intelligent Systems and Signal Processing*, Volume 1, pages 339-346, October 8-13, 2003.
- [14] Y. Li, T. Callahan, E. Darnell, R. Harr, U. Kurkure, and J. Stockwood. Hardware-Software Co-Design of Embedded Reconfigurable Architectures. *Design Automation Conference*, June 2000.
- [15] M. Baleani, F. Gennari, Y. Jiang, Y. Patel, R. K. Brayton, and A. Sangiovanni-Vincentelli. HW/SW Partitioning and Code Generation of Embedded Control Applications on a Reconfigurable Architecture Platform. In *Proceedings of the Tenth International Symposium on Hardware/Software Codesign*, May 2002.
- [16] J. Fleischmann, K. Buchenrieder, and R. Kress. Codesign of Embedded Systems Based on Java and Reconfigurable Hardware Components. *Design Automation and Test in Europe*, March 1999.
- [17] R. Niemann and P. Marwedel. An Algorithm for Hardware/software Partitioning using Mixed Integer Linear Programming. *Design Automation of Embedded Systems*, Vol. 2, No.2, pp.165-193, 1997.
- [18] D. Saha, R. S. Mitra, and A. Basu. Hardware/software Partitioning using Genetic Algorithm. In *Proc. of 10th Intern. Conference on VLSI Design*, 1997, pp.155-160.