

An Agent-Based Architecture on Reconfigurable System-on-Chip for Real-Time Systems

Yan Meng

Department of Electrical and Computer Engineering
Stevens Institute of Technology, Hoboken, New Jersey 07030, USA
Email: yan.meng@stevens.edu

1. Introduction

The definition of real-time system is any system that is both logically and temporally correct. Logically correctness means the system satisfies all functional specifications. Temporal correctness means the system is guaranteed to perform these functions within explicit time frames. Real-time systems are classified as hard or soft. Hard systems have catastrophic consequences if the temporal requirements are not met – up to and including complete system destruction. Conversely, soft systems only have degraded performance if the temporal requirements are not met.

Conventionally, all of the tasks of a complex real-time system are implemented on a general purpose processor in software due to its great flexibility, where a powerful processor is required to guarantee that hard deadlines can always be met, even for very rare conjunctions of events. Sometimes the sensors of a real time system are supposed to work in parallel and independently, for example, in a mobile robot system, it would be impossible to implement the parallel sensing in software due to its inherent sequential processing. 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 operate in parallel, concurrently, and asynchronously, 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.

Reconfigurable computing is intended to fill the gap between hardware and software, achieving potentially much higher performance than software, while maintaining a higher level of flexibility than hardware. Field-programmable gate arrays (FPGAs) is one of the most popular reconfigurable devices with various configurable elements and embedded blocks providing new solutions for high density and high-performances embedded system design. These platforms not only enable system architects to design and develop complex custom systems using embedded processors and interoperable IP cores, but also provide technologies such as dynamic reconfiguration. The advances of FPGA technology make the hybrid hardware-software architecture of reconfigurable system-on-chip (rSoC) feasible in implementation, which can provide both runtime flexibility and real time performance for complex applications. These two features are not simultaneously achievable by traditional pure software or hardware implementations.

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 system-on-chip (rSoC) architecture. This rSoC architecture raises the possibility of different hw/sw (hw/sw) codesign approaches. To speed up the system integration and hw/sw codesign process on a rSoC platform, a unified hw/sw interface is necessary. Based on this rSoC platform, a *Belief-Desire-Intention (BDI)* agent model is proposed in this project as a unified structure for both hardware and software, which is intended to design high-level aspects of systems and to simplify the hw/sw partitioning and communication. The system is first decomposed into 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 based on the heuristic approaches. Compared to the regular module-based architecture, the BDI agent model can provide more flexibility, intelligence, autonomy, and scalability.

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 developing different transport-dependant adapters to new systems.

This rSoC platform is 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, it can be effectively managed by a dynamic reconfigurable hardware logic units. *Dynamic reconfiguration* is based on the idea that parts of the system remain available during the reconfiguration. Although disruption is unavoidable, the impact of the disruption should be minimized, as well as the duration of the effects of this disruption. The reconfiguration mode should introduce minimal overhead during normal operation. Deploying dynamic run-time reconfiguration in systems results in reduced chip area and power consumption. Therefore, several hardware reconfiguration modules are proposed in this project.

To evaluate the real time performance of the proposed agent-based architecture on a rSoC platform, two case studies are presented, one is a mobile robot system, and the other is a mobile vision system for people detection and tracking. Vision sensors are the powerful sensors for a mobile robot for situation awareness and target detection. However, most computer vision approaches have underlying assumptions, such as large memory, high computation power, static vision system, or off-line processing. When a vision system is on a mobile platform, there are some constraints which are specific compared to the static vision system. First, the system has to be highly robust to adapt to the changing illumination and environmental structures. Second, the real-time processing is very important to realize a natural reactive behavior and prevent any serious damages. Furthermore, a mobile robot system consists of not only a vision system but also the navigation and control systems. Most developed mobile robot systems are designed to work efficiently under some certain environments, such as structured indoor buildings, or outdoor open environments. However, in some emergency response situations, such as in an urban search and rescue (USAR) environment or some unknown hazardous environments, the working environment of the mobile robot is unpredictable and dynamically changed. All of these tasks have to be processed by the on-board processor of a robot under the real-time constraints. , A software-only solution will push the limite of the processing capability, which may lead to a very conservative computation solution and a very challenging real-time scheduler. Furthermore, with one or multiple microprocessor(s) in the system, it is difficult to handle the high frequency of the external I/O. The overhead for the interrupts reduces the microprocessor performance. Therefore, the reconfigurable system-on-chip (rSoC) platform is applied in this project to tackle these issues, which could respond with greater flexibility, processing capacity, and performance for most real-time systems, especially for those with dynamic environments and may encounter various emergencies. .

2. Related Work

Most previous work on reconfigurable embedded platform for real-time systems is implemented on software only. Gafni et al. [1] proposed an architectural style for real-time systems, in which the dynamic reconfiguration was implemented by a synchronous control task in response to the sensed conditions. To handle run-time dependencies between components, Almeida et al. [2] employed 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. [3] presented a hierarchically self-adaptation model for a robot system to provide fault-tolerance. MacDonald et al. [4] proposed some design options for dynamic reconfiguration with their service-oriented software framework. Stewart et al. [5] proposed 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. [6][7] presented a self-reconfigurable robot design with a distributed software architecture that followed 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 [8][9]. In [10], the system used FPGAs that were dynamically reconfigured on Dynamically Programmable Gate Arrays at runtime to implement different functions. The fastest solution for reconfiguration was through context switching which was able to store a set of different configuration bitstreams and make the context switching in a single clock cycle. [11] [12] proposed contest switching FPGA architectures which emphasized on both proving fast context switching as well as fast random access to the configuration memory. This is important because you may want to change one or more of your configuration streams inside the FPGA at runtime.

To fully take advantage of the reconfigurable characteristics of FPGAs, some self-reconfiguration mechanisms were proposed. Blodget et al. [13] proposed a self-reconfiguration platform for embedded system using ICAP and an embedded processor core within the FPGA. Fong et al. [14] developed a system that used the RS-232 port to transfer configuration bitstreams to the FPGA, which may easily lead to a bottleneck for the transfer speed and an increase the reconfiguration latency.

As the soft processor cores for FPGAs become available in the rSoC platform, the hw/sw codesign issues becomes critical in the overall system design procedure. Some of the more recent hw/sw codesign research used reconfigurable processors as the platform for implementation [9][15], where their architectures combined 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 [16][17]. In [17], the hw/sw partitioning problem was modeled as a Constraint Satisfaction Problem (CSP), and a genetic algorithm based approach was proposed to solve the CSP to obtain the partitioning solution.

The major innovative architecture design proposed in this project is applying the multi agent paradigm to the reconfigurable embedded system on both software and hardware design, and establishing a framework by which hardware agents can be dynamically reconfigured to meet the system's real-time constraints. The proposed reconfigurable architectures can provide the consistency preservation, low reconfiguration latency, and fast random access to the configuration memory. Due to its generality, the proposed architecture can be extended to large real-time applications.

3. A BDI-Based Multi-Agent Architecture

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 assigned. 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. In the agent model, as shown in Fig. 1, beliefs and desires influence each other reciprocally. Furthermore, beliefs and desires both influence intentions. The model includes three external ports: inter-agent communication, control, and input/output. The control port is 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 to/from other agents and cooperate with others.

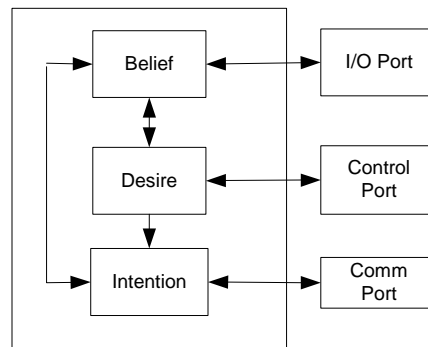


Fig.1. The BDI agent model

The agent control loop is: first determine current beliefs, goals and intentions, find applicable plans, then decide which plan to apply, and finally start executing the plan. Both the software agents and hardware agents in our system adopted the BDI architecture model. The BDI agent can be expressed in a structure as followings:

```
<Agent>{<Beliefs>
    Constraints; Data Structures;
    <Desires>
    Values; Condition; Functions;
    <Intentions and Commitment>
    Methods; Procedures;}
```

This structure can be implemented by high-level software languages on microprocessors, or Verilog hardware description language (VHDL) on FPGA. The BDI agent is capable of providing the following features which are impossible for the regular module-based architecture.

- **Autonomy.** Once launched with the information describing the bounds and limitations of their tasks, BDI agents are able to operate independently of and unaided by their use.
- **Social ability.** To effectively change or interrogate their environment, BDI agents possess the ability to communicate with the outside world through their input/output ports, and communicate with other agents through inter-communication ports.
- **Reactivity.** BDI agents are able to perceive their environment and respond to changes in a timely fashion.
- **Proactivity.** To help BDI agents to be adaptive to new situations, they are able to exhibit proactivity, that is, the ability to effect actions that achieve their goals by taking the initiative.

Some agent-oriented implementations have been developed, such as JACK [18] and distributed multi-agent reasoning system (dMARS) [19]. 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 its internal state. 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 executes 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 high-level software languages, while in hardware we have to specifically define all of the hardware agent entities, their associate ports and parameters in VHDL. The hardware agent interface should be simple for customization to any specific applications without significant rework.

4. Hardware/Software Codesign

Hw/sw codesign 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. Fig.2 shows the codesign flow performed in order to transform a system-level specification of a mixed hw/sw implementation. The codesign process is composed of four steps: system-level specification, system-level partitioning, communication synthesis, and software and hardware synthesis. The goal of hw/sw codesign is to produce an efficient implementation that satisfies the performance and minimizes the cost, starting from the initial specification. The mixed hw/sw system will be mapped from system behavior description.

The proposed agent-based architecture provides a unified model for hw/sw co-simulation and co-synthesis. To achieve high performance of overall system, an effective hw/sw partitioning methodology and a competent communication protocol among the agents are required.

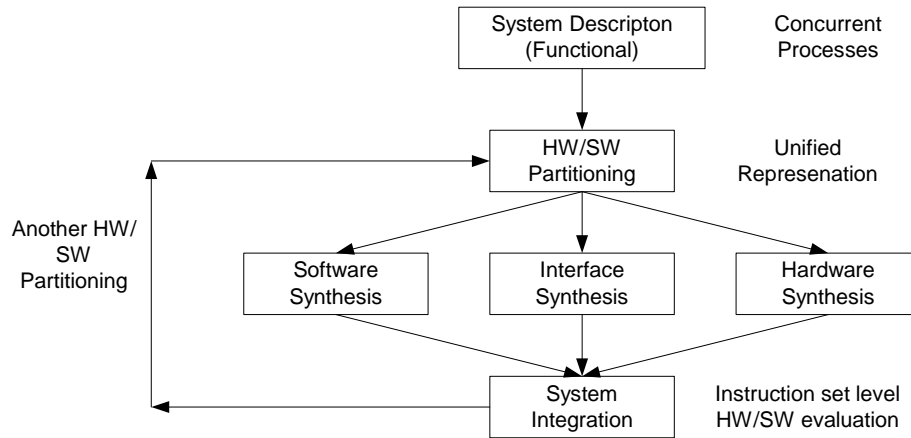


Fig.2. Typical HW/SW codesign process

4.1. HW/SW Partitioning

It is desirable to propose a method which can automatically partition the agents into hardware and software on the fly based on the task parameters of each agent, such as execution time, release time, and deadline. However, 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. Furthermore, the dynamic partitioning becomes more complex if it has to handle some sporadic tasks due to environmental changes.

In this project, since our research focus is on the architecture design, an off-line heuristic partitioning method is applied to simplify the hw/sw codesign procedure. In general, the following partitioning policies are required to follow. First, 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. Second, due to the different implementation platforms of hardware and software, the agents who have heavy communication or dependencies should be all partitioned either to hardware or software to minimize the communication costs. Third, the agents with hard deadlines are distributed to hardware, while the agents with soft deadlines may be implemented in software.

4.2. Multi-agent Communication Mechanism

Agents are generally designed with a specific purpose in mind. They can 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 may 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 intension 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) [18], and Foundation for Intelligent Physical Agent (FIPA) [19]. 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 have also 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, which leads to a new on-demand message passing (ODMP) protocol proposed in this project. A typical example of two agents connected by a communication path is shown in Fig.3. In this mechanism, each agent creates a message queue 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 an FIFO order.

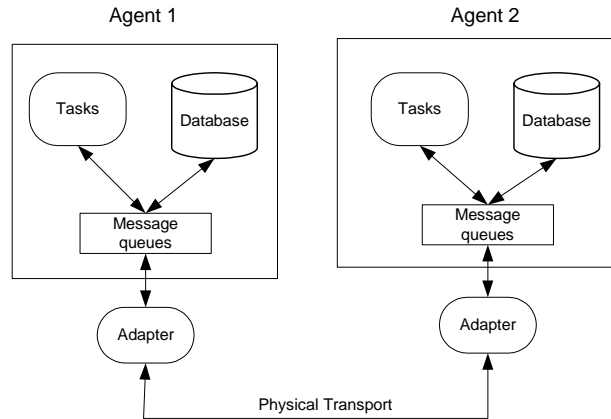


Fig.3. Multi-agent communication structure

Each agent may have multiple current tasks need to be implemented. To reduce the routing complexity of hardware agents and memory requirement, the database only maintains the list of all other related agents in the system along with their current status. When we say “related”, we mean the agents who need to receive/send the message from/to the current agent. The current status of the agents would help to improve the fault tolerance. 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 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.

To provide a media-independent agent communication mechanism, an *adapter* mechanism is proposed. One adapter is designed to be associated with each different physical transport, which can be a bus driver for software agent, or other high-level communication protocols, such as UDP or Ethernet. The adapter mechanism provides a uniform interface for the agents, making the agent to be independent from the specific transport being used. One adapter may be connected to multiple agents if necessary. When an agent needs to send a message to other agents, it passes the message buffer to the adapter first, then the adapter sends data out through the physical transport. Similarly, when an agent receives a message from other agents, the 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 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 adapters. 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. Service header usually identifies the internal service sending the message and the message type, which are application dependant.

When an agent sends a message to a remote agent, this data can be a fragment of the whole message if the total message size exceeds the size of 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.

5. Hardware Agent Reconfiguration Modules

Performing reconfiguration on a running system is an intrusive process. Reconfiguration may imply, for example, interference with ongoing interactions between entities. One of the main issues of dynamic reconfiguration is consistency preservation [20]. Changing management functionality must assure that system parts that interact with entities under reconfiguration do not fail because of reconfiguration, i.e., system consistency needs to be preserved. Furthermore, the runtime reconfiguration latency can be

significant. To maximum application execution performance, such loading overhead must be minimized. In this section, various techniques are proposed to reduce/tolerate the configuration latency.

The Virtex-II Pro device is used as the FPGA platform in our project, which is capable of implementing flexible, high performance, and low-cost system-on-chip designs by combining a variety of features embedded in the FPGA fabric with specially developed hw/sw IP cores [21]. Particularly, it cooperates fully embedded IBM PowerPC 405 processor core [22]. The embedded PPC 405 core is a 32-bit Harvard architecture processor with functional units such as cache unit and memory management unit (MMU). It is capable of more than 300 MHz clock frequency and 420 Dhrystone MIPS, and is contained in a processor block. The processor block also contains on-chip memory controllers and integration circuitry compatible with IBM CoreConnect bus architecture that enables the compliant IP cores to integrate with this block.

5.1. Virtual Agent Reconfiguration Module

The consistency preservation could become a very complicated problem. In order to preserve the consistency while keep the dynamic reconfiguration procedure simple, we make the following assumptions for the reconfiguration models.

- 1) Agents work independently.
- 2) Agents can communicate with other agents.
- 3) Each agent can have a finite number of schemes which is a description of possible events. Each scheme of the agent can be created a priori before the application is implemented, and can be stored in the external memory.

Based on the above assumptions, a *Virtual Agent Reconfiguration (VAR)* module is proposed, which is based on the work of Sckanina et al. [12] by implementing a user defined FPGA inside an ordinary FPGA. As shown in Fig. 4, this method consists of a number of *virtual* states that each agent could possibly go through at runtime. These virtual states are defined as configuration bitstreams which are loaded into the reconfigurable hardware memory when the system starts up. Thus, in this scheme the FPGA configuration registers are fixed at runtime. The device is reconfigured only by simply selecting an active state by a multiplexer based on its local point of view. There is no reconfiguration latency due to the static reconfiguration. If the device contains plenty number of agents, the cost of this method is extremely high and the efficiency is very poor because all possible agent states must be implemented which requires more reconfiguration resources. However, the number of logic blocks in the Vertex blocks is large (for example, XC2VP30 has 30k logic cells) and each block can be configured in a set of ways. Therefore, if the number of agents in a real time system is relatively small, and the computation of data processing is not extremely intensive, this reconfiguration method would be suitable due to its low overhead and low reconfiguration latency.

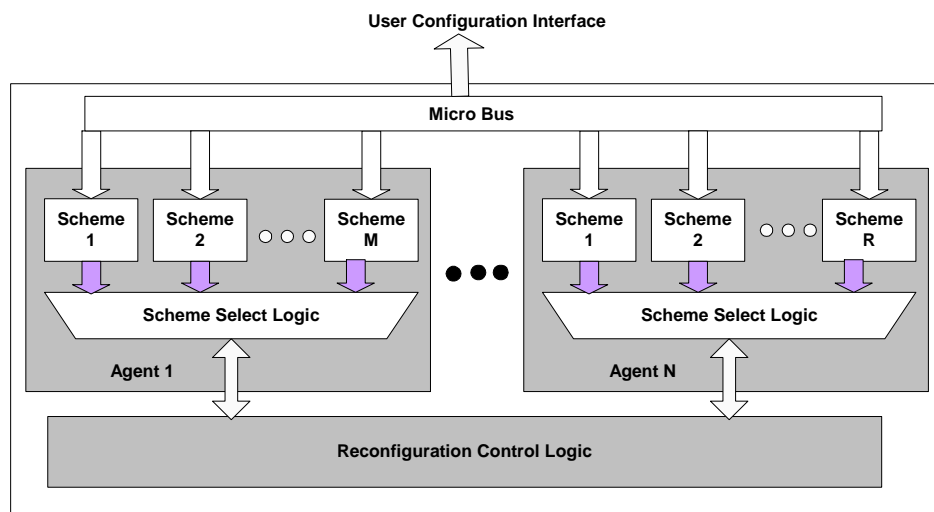


Fig. 4. Implementing a virtual agent reconfiguration

5.2. Pipeline Agent Reconfiguration Module

In order to use the reconfigurable resources more efficiently when the number of reconfigurable agents is large, one option is that the replacement agent reconfiguration overwrites the previous agent state while suspending the functionality of the reconfigurable logic. This mode supports several temporal modes of operation. However, there is high reconfiguration latency because the system must spend lots of time to load the selected agent states from high capacity/low cost hardware configuration storage to the partially reconfigurable logic. This method is only suitable for those applications that do not have critical real-time constraints.

Instead of waiting for the long latency time of loading configurations from low-cost hardware to the reconfiguration memory, we propose the *Pipelined Agent Reconfiguration (PAR) module*. First, the FPGA is partitioned into a set of static and dynamic sub-FPGAs, where each sub-FPGA can be configured separately using partial reconfiguration. The static sub-FPGA is fixed and always active at runtime, and is used to manage the dynamic sub-FPGAs which are configured at runtime and are not always in operation. That is, only one of the dynamic sub-FPGAs is active at any given time. Thus, the one which is not active can load its new configuration through reconfiguration control logic by partial reconfigurations, while the other one is executing. In this way, the FPGA becomes a pipelined unit where the configuration is pipelined with execution, which is shown in Fig. 5. The number of dynamic sub-FPGAs could be increased if the execution time is shorter than the configuration time. By applying this pipelined method, the reconfiguration latency is greatly diminished, which is critical for the real-time systems. However, only part of the reconfigurable resources can be used at a given time. This method can obtain the optimized balance between the reconfiguration latency and resource occupancy.

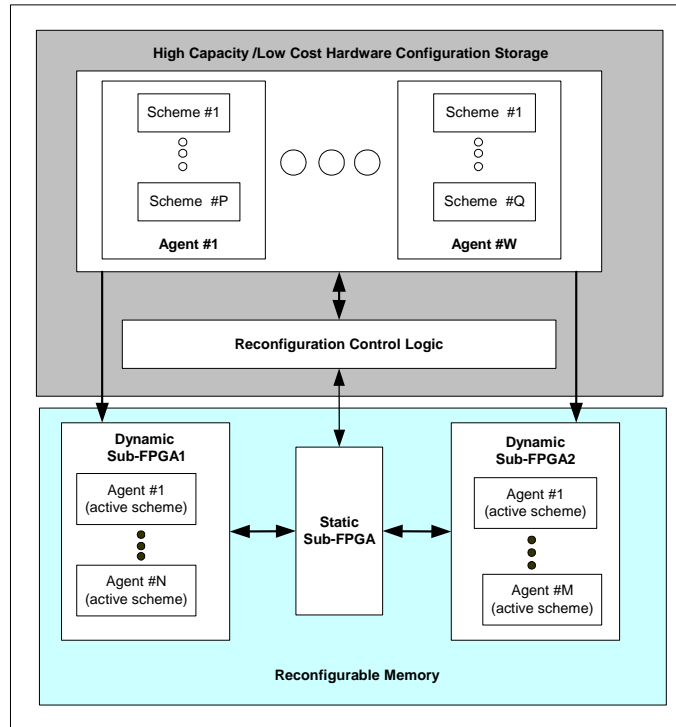


Fig. 5: Pipelined redundant reconfiguration architecture

5.3. A Self-Reconfiguration Module using Virtex-II Pro FPGA

The above VAR and PAR modules can be applied to any partially reconfiguration platform. Since Virtex-II Pro FPGA is adopted in our project, a self-reconfiguration method is proposed based on Virtex-II Pro FPGA. *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. Since the control logic is located as close to the logic array as possible, the latencies associated with accessing the configuration port is minimized.

The Virtex-II Pro configuration architecture features an Internal Configuration Access Port (ICAP) that provides the user logic with access to FPGA configuration interface and therefore access to memory bits of configuration memory [23]. In active partial reconfiguration, new data can be loaded through ICAP to dynamically reconfigure a particular area of FPGA while the rest of FPGA is still optional. Some self-reconfiguration platforms proposed in [13] [14] 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.

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 Fig. 6, the fixed logic includes common fixed logic circuits, ICAP, configuration controller, multiple dual-port block RAMs (BRAMs), embedded processors, and external memory. The embedded processors provide intelligent control of the device reconfiguration at runtime. The embedded processors communicate with FPGA logic through IBM-designed CoreConnect bus architecture. The DDR external memory is connected to the processors through process local bus (PLB bus).

The fixed logic part is located at 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.

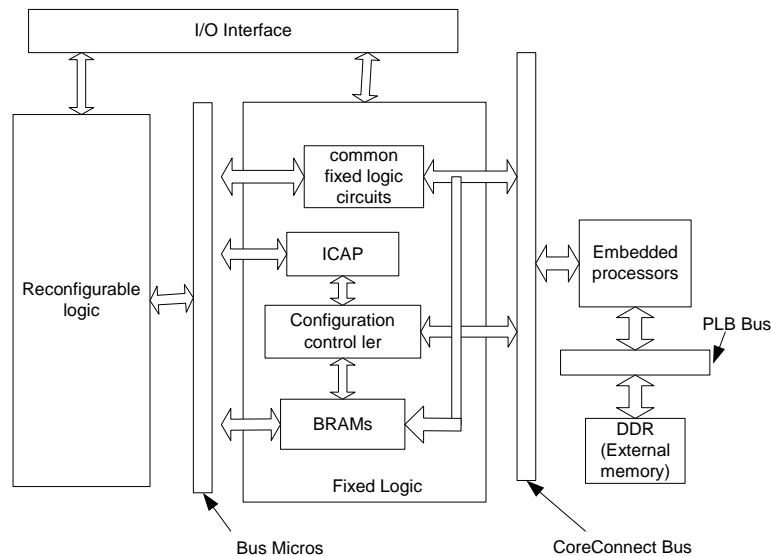


Fig. 6. A self-reconfiguration platform

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 processors, acting like a shared memory between the processors and reconfiguration logic part. 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 processors.

To improve the system robustness, two trigger conditions for the dynamic reconfiguration are applied here, i.e., bottom-up and top-down. The bottom-up method is through event interrupts invoked by the some specific agents. The top-down method is that the processors start the reconfiguration due to decisions based on the input information.

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 through CoreConnect Bus directly if it is fixed hardware agent. The message can be stored on the DDR SDRAM via CorConnect PLB bus.

6. Case Study

6.1. The Prototype Platform

A Pioneer 3DX mobile robot system from ActivMedia Robotics is adopted as a mobile vehicle in our project, as shown in Fig. 7(a), which 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 SONY CCD camera, one laser rangefinder and eight SONARs for range measurement, and two motor encoders for odometry measurement.



(a) Pioneer 3DX robot



(b) ML310 embedded platform

Fig.7. The experimental platform

The Xilinx ML310 embedded development platform, is a Virtex-II Pro based platform suitable for rapid prototyping and system verification, and is installed to the robot as the 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, FPGA UART, standard PC I/O ports, and high speed I/O through RocketIO Multi-Gigabit Transceivers (MGTs). 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 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++ language.

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 queues are saved on internal BRAM for hardware agents and external DDR SDRAM for software agents. The self-reconfigurable module is applied for dynamic reconfiguration of hardware agents in our experiment.

6.2. A Mobile Robot System

To evaluate the proposed architecture, we develop the agent-based architecture for a mobile robot navigation system [24]. The block diagram is shown in Fig. 8. Usually one mobile robot system has multiple sensor systems, where each sensor agent is designed to acquire some specific information from the external environment.. All of the sensor agents work parallel and independently. Sensor fusion agent analyzes the sensor information acquired by all of the sensor agents, fuse the sensor data if necessary, and send the fused sensor data to the higher-level processing agents. Global path planning agent plans an optimized path for the robot to navigate across the global environment given a starting point and a destination point. Obstacle avoidance agent detours the robot when there are obstacles in the way to prevent the robot hitting the obstacle.

Actuator management agent receives requests from processing agents, and distributes the control commands to individual actuator agents. Human-robot interface agent receives the events or processed environmental information from sensor fusion agents, and then sends corresponding commands to the robot. Emergency stop agent usually takes commands from human and sends request to actuator manager agent to stop the robot.

According to the hw/sw codesign technology, the hardware and software agents are partitioned as the follows: the sensor agents, sensor fusion agent, actuator management agent, and actuator agents are configured in hardware on FPGA, while all of the processing agents, including path planning, obstacle avoidance, emergency stop, and human-robot interface agents are implemented in software on embedded processor core.

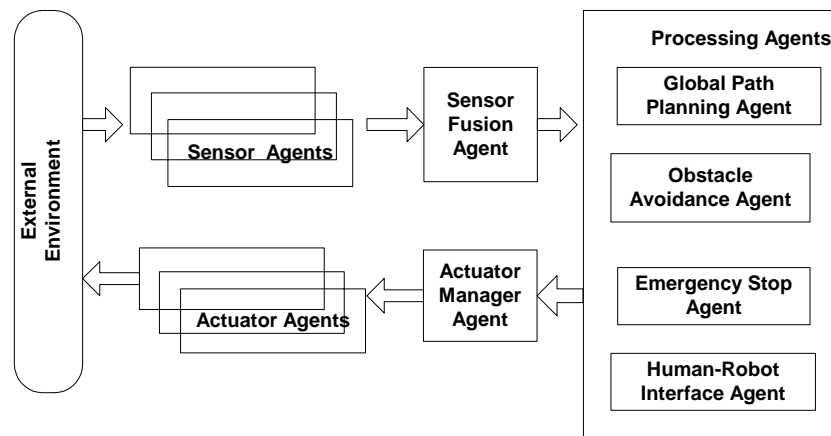


Fig. 8. Multi-agent architecture for a mobile robot navigation system

Based on the above multi-agent architecture, when a mobile robot navigates to different environments or sensors encounter some malfunctions, only affected sensor agents need to be reconfigured, while the other agents will continue their own tasks without any influence. The reconfiguration can be either triggered by the sensor agents by invoking an interrupt to processor, or by the processor itself based on its data checking algorithm.

The real world scenario is shown in Fig. 9. The mobile robot (circle) needs to navigate itself from one fixed starting point at room B with to the fixed destination point at room A while searching a specific feature target (a green cube) and avoiding all of the obstacles on its way. A simple color detection algorithm is applied to the vision system to detect the target. The map of two office rooms is installed to the robot before the navigation, and the obstacles (triangle and oval) are randomly scattered.

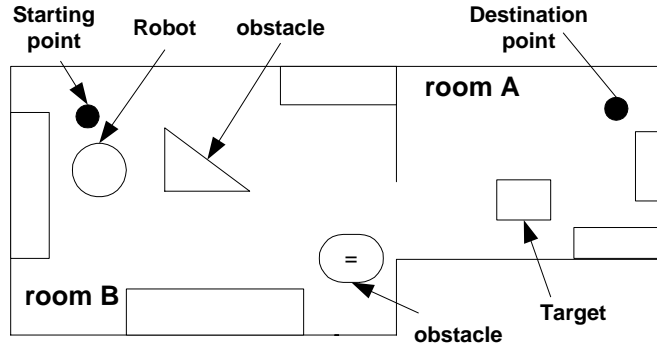


Fig.9. The navigation environment

By implementing both the hw/sw codesign and software only approaches to the mobile robot, the robot has successfully navigated itself to the destination point, and found the target without hitting any obstacles on its way. Five configurations with different target locations in room A and different obstacles distributed in room B are designed in the above environment. 10 runs have been conducted on each configuration for both hw/sw and software-only approaches. The average navigation time of experimental results are shown in Fig. 10. The resource usage of hardware agents is listed in Table 1.

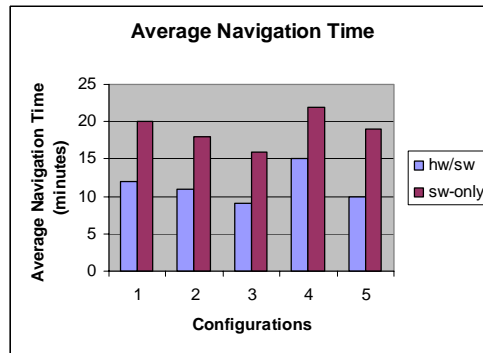


Fig. 10: Average navigation time comparison for real-world experiments

Table 1: Resource Usage of Hardware Agents

Resources	Self-Reconfiguration Model
IOs	56.34%
Function Generators	73.34%
CLB slides	65.32%
DEEs or Latches	5.23%

It can be seen from Fig. 10 that there is a dramatic decrease in average navigation time in a dynamic environment with hw/sw codesign comparing with sw-only approach. The hw/sw 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.

Since only the simple color-based target detection is applied in our experiment, we didn't take fully advantage of the proposed hw/sw architecture. For the more complex vision-based robots, such as service robots, the overall system responsiveness should be improved significantly using the proposed platform

because the vision-related algorithms are extensive time consuming with software-only approach. Therefore, a mobile vision system for people detection and tracking is discussed in next section.

6.3. A Mobile Vision System for People Detection and Tracking

Some researches have been conducted for people detection systems using one single cue at the detection, such as motion [25], color [26], contour [27], or face's features (such as eyes, nose, and mouth, etc.) [28]. Due to the natural, complex, and dynamic working environment of a mobile robot, the robot vision system has to be highly robust and independent with the application scenario. Therefore, some researches proposed the combinations of multiple cues for people segmentation [29] [30] [31]. In this project, we propose a detection method combining the color and motion cues.

First, skin-color classification is applied to find faces in images since it is independent from ego-motion of the camera systems. To represent skin color, the dichromatic r-g-color space ($r=R/(R+G+B)$, $g=G/(R+G+B)$), which is normalized in brightness and thus is widely independent from variations in luminance. To improve the system real-time performance, we have built up a look-up table with manually classified skin color pixels in the r-g-color space for 30 people under different illumination conditions.

Then, motion detection is applied to distinguish the people moving in the scene from the stationary background. For a stationary camera system, motion detection can be conducted by subtraction of two successive image frames. However, if the camera system is moving, the ego-motion has to be taken into account. The algorithm proposed in [32] is applied to estimate the compensation for the ego-motion of the mobile robot. Assume the robot motion parameters can be obtained from the navigation and control systems. The next successive image can be predicted by using the epipolar transformation. The predicted image is matched with the actual image from the scene, and all non-matching areas represent the moving objects.

It may occur frequently that multiple Region of Interests (ROIs) are detected in one image, where a fusion algorithm is necessary to merge those ROIs. To simplify the merging procedure, we assume that there are relatively small changes in size and position of people from the successive image frames, in other words, the people is moving in a relatively slow speed. Then a probabilistic approach based on the color and motion cues as well as the geometric parameters of the object in the images is applied to select the most likely ROI.

The condensation tracking algorithm in [33] is adopted here for face tracking. The task of calculating the probability of the presence of a face for every pixel and tracking the resulting density function over time is solved by an approximation of the density function. Then, a recursive filter is applied to update the density function on each sample. Once a person is tracked, the samples of the condensation algorithm are concentrated on his/her face.

For a mobile vision system, a large amount of images from vision system need to be processed under real-time constraints. Image matching is critical for navigation system since 3D environment knowledge can only be obtained through multiple views. With the partial reconfiguration features of FPGA, it is possible to run the image loading and image matching simultaneously. This parallel scheme can reduce the implementation time dramatically since the image loading from external memory to reconfigurable hardware is usually extremely computational extensive. When the image is loaded onto the buffer from I/O, the other buffer is applied for high-speed matching algorithm. When the image loading is finished, the control logic will check the status of another buffer. If the matching is done, the two buffers exchange operations. Otherwise, the control logic will inform the loading buffer to wait till the other buffer finishes the matching, and then exchange the operations.

Furthermore, traversing the image is necessary to identify the Region of Interests (ROIs). To speed up this traversing procedure, each loaded image can be divided into several regions, where each region can be assigned to different FPGA logic block. A separate control logic is responsible for whole image information integration. By paralleling the searching procedure, the overall speed of the image processing can be significantly reduced. Since the FPGA blocks are limited for each rSoC platform, the dynamic reconfiguration is conducted so that the new image regions will be continuously loaded into each block after the previous block has been processed. If some of the regions finish their traversing faster than others, it is possible that new image regions can be loaded in the finished regions to start processing. Therefore, a pipeline scheme can be formulated to improve the processing efficiency by using the FPGA logic blocks greedily.

To evaluate the proposed architecture, we develop the agent-based architecture for a mobile vision [34], and the block diagram is shown in Fig.11, where rectangles represent hardware agents and ovals represent software agents. Since robot path planning agent and robot/human interface agents are complex and not critical in real-time reaction, they are partitioning into the software agents, where all of other time-critical and extensive time consuming agents, such as image processing, target detection and tracking, camera control, laser and sonar sensor, sensor fusion, and robot actuator control, are assigned to hardware agents. Unlike the other hardware agents which are located on the reconfigurable logic blocks, the reconfiguration control agent locates on the fixed logic block, which needs to control the reconfiguration procedure of the other hardware agents.

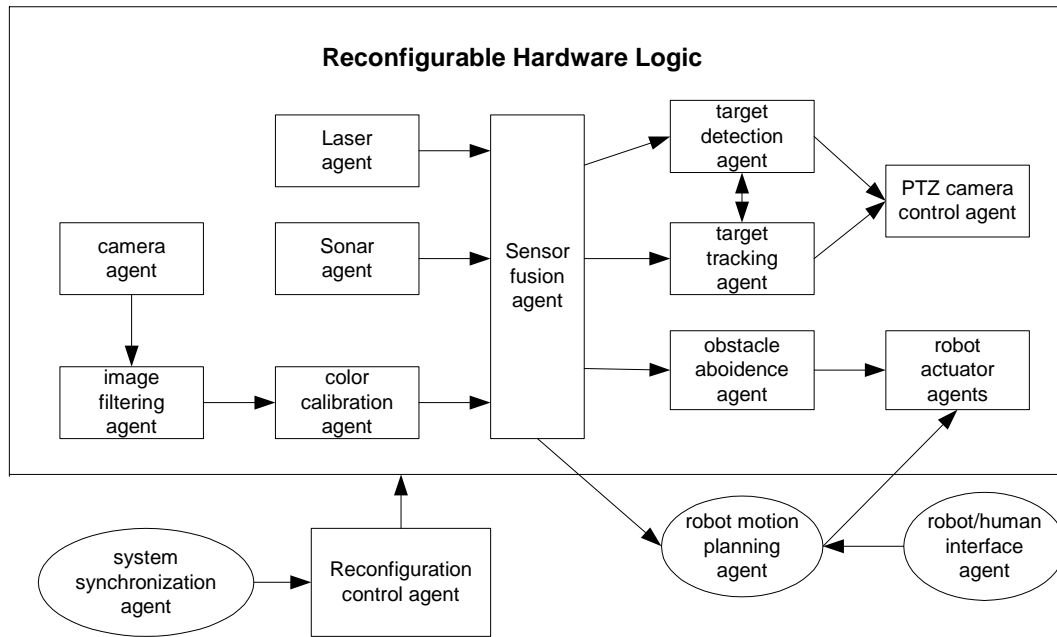


Fig.11. Agent-based architecture for a mobile vision system

First, the camera captures the image from the environment, and sends the images to the image filtering agent. After the color calibration agent finishes its processing to remove the illumination affects, the sensor fusion agent combines the image information with the laser agent and sonar agent. The output of the sensor fusion agent can serve for target detection agent, target tracking agent, obstacle avoidance agent, and robot motion planning agent. The target detection and tracking agent will control the movement of the PTZ camera to keep the target inside its field of view. It is worth noting that at any given time, the obstacle avoidance agent and motion planning agent can work in a complementary mode, which means that only one agent can work at a time. If obstacles are detected, obstacle avoidance agent works, otherwise, motion planning agent works. The motion control signal will be sent to robot actuator agents. Since the target detection and tracking agents are all configured in the hardware FPGA, they can be implemented concurrently with the obstacle avoidance agent, which means that the vision system can detect and track the target while avoiding the obstacles on its way. In this project, we use people as our target for detection and tracking by the mobile vision system.

To evaluate how much performance to be gained by the hw/sw codesign platform compared to the software only platform, both software-only and hw/sw co-design approaches have been implemented for face detection and tracking using the vision system installed on the mobile robot. The method in [31] was adopted in the experiments for face detection due to its high detection rate and very low false positive rate. The detection rate is 61.83% and false positive rate is 0.0004%,

The experiments for both approaches have been conducted for 20 times in terms of different initial conditions of the vision system. To make the experimental results to be more comparable, for the same initial condition, both the software-only and hw/sw approaches are implemented respectively. One person is sitting on a chair at a fixed location facing to the direction where the robot locates, where the robot

initially starts from different locations with different facing direction of the camera system. There are some obstacles dynamically sitting on the way between the vision system and the person. Before the robot detects the face and tracks the face, since the robot has no idea where the person is, therefore, random movements are initiated until the vision system detects the face, then the robot moves towards the face while controlling the PTZ camera to tracks the face afterwards.

The detection time is defined as the time difference from the initial time to the time when the face is detected. The experimental results are shown in Fig. 12. The average detection time for software-only is 33.05msec, while the average time for hw/sw co-design approach is 8.375msec. It is obviously that the co-design approach can improve the system performance significantly in its responsiveness and fault tolerance for real-time systems, especially in the situations where the environments are dynamically distributed by some obstacles.

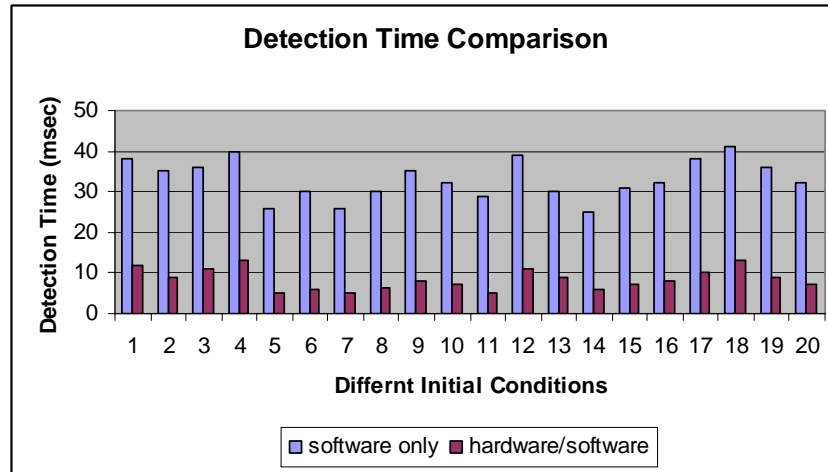


Fig. 12. Detection time comparison

7. Conclusion and Future Work

This paper proposes an agent-based architecture on a rSoC platform for real-time systems. The unified BDI agent structure significantly simplifies the hw/sw partitioning and communication. Some heuristic hw/sw partitioning methods are developed as the first step for hw/sw codesign. Then a novel ODMP protocol is proposed for the agent-to-agent communication. By developing different adapters for physical transports, the agent-based architecture design can be transport independent. To take fully advantage of the partial reconfiguration provided by FPGA platforms, some reconfiguration schemes are proposed to provide the consistency preservation, low reconfiguration latency, and fast random access to the configuration memory.

To evaluate the proposed agent-based architecture on rSoC platform, two real-time systems have been conducted using P3-DX robot with vision systems. First system is a mobile robot system with simple detection capability, and the second one is a mobile vision system for people detection and tracking. The experimental results show that the proposed architecture is feasible and much more efficient compared to the software only approach. Due to its generality and flexibility of the proposed agent-based architecture, it can be easily extended to large various real-time applications

The proposed BDI agent-based architecture provides a very flexible platform for the future extension, such as learning capability and proactivity to adapt to the dynamic environments. Our future research will focus on dynamic hw/sw partitioning and scheduling instead of just using an off-line heuristic method. In addition, we will also investigate the learning capability of the BDI agent architecture to make the overall system be more robust and adaptive to the environmental changes.

References

- [1] V. Gafni. Robots: a real-time systems architectural style. In *Proc 7th European software engineering conference*, pages 57-74, Toulouse, 1999.
- [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] J. Cobleigh et al. Containment units: a hierarchically composable architecture for adaptive systems. *ACM SIGSOFT Software Engineering Notes*, 27(6):159-165, November 2002.
- [4] 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.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] Y. Meng. A Dynamic Self-reconfiguration Mobile Robot Navigation System. *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2005)*, 2005.
- [9] 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.
- [10] 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.
- [11] S. M. Scalera and J.R. Vazquez, “The design and implementation of a context switching FPGA”, *IEEE Symposium on FPGAs for Custom Computing Machines*, 1998, pp. 78-85.
- [12] L. Sckanina and R. Ruzicka, ‘Design of the special fast reconfigurable chip using common FPGA’, in *Proc. Of Design and Diagnostics of Electronic Circuits and Systems – IEEE DDECS’2000*, 2000, pp.161-168.
- [13] 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.
- [14] 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.
- [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 Hw/sw Codesign*, May 2002.
- [16] R. Niemann and P. Marwedel. An Algorithm for Hw/sw Partitioning using Mixed Integer Linear Programming. *Design Automation of Embedded Systems*, Vol. 2, No.2, pp.165-193, 1997.
- [17] D. Saha, R. S. Mitra, and A. Basu. Hw/sw Partitioning using Genetic Algorithm. In *Proc. of 10th Intern. Conference on VLSI Design*, 1997, pp.155-160.
- [18] Agent-Oriented Software Pty Ltd, JACK Manual (v4.1), www.agent-oriented.com, 2003.
- [19] 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, pg. 155-176, Berlin, 1998.
- [20] J. Kramer and J. Magee, “The Evolving philosophers problem: dynamic change management”, *IEEE Trans. Software Engineering*, 16(11):1293-1306, 1990.
- [21] “Virtex-II Platform FPGA Handbook”, version 2.0, Xilinx, Inc., 2004.
- [22] “PowerPC Processor Reference Guide”, version 2.0, Xilinx, Inc., 2003.
- [23] “Virtex-II Platform FPGA User Guide”, version 4.0, Xilinx, Inc., 2005.

- [24] Y. Meng, An Agent-Based Reconfigurable System-on-Chip Architecture for Real-time Systems, *The 2nd International Conference on Embedded Software and Systems (ICCESS 2005)*, Xian, China, December 16-18, 2005.
- [25] K. Rohr, "Towards Model-based Recognition of Human Movements in Image Sequences", *CVGIP: Image Understanding*, vol. 59, no.1, 1994, pp.94-115.
- [26] Y. Raja, S. J. McKenna, and S. Gong, "Tracking and Segmenting People in Varying Lighting Conditions," in *Proc. 3rd Int. Conf. on Automatic Face and Gesture Recognition*, pp. 228-233, 1998.
- [27] J. Davis and V. Sharma, "Robust Detection of People in Thermal Imaging," *17th International Conference on Pattern Recognition (ICPR'04)*, 2004, pp. 713-716.
- [28] H. A. Rowley, S. Baluja, and T. Kanade, "Neural Network-based Face Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1), 1998, pp.23-38.
- [29] S. Feyrer and A. Zell, "Detection, tracking, and pursuit of humans with an autonomous mobile robot", *Proceedings of the International Conference on Intelligent Robots and Systems (IROS'99)*, 1999, pp. 864-869.
- [30] B. Froba and C. Kublbeck, "Face detection and tracking using edge orientation information", *SPIE Visual Communications and Image Processing*, 2001, pp. 583-594.
- [31] P. Viola and M. Jones, "Robust real-time object detection", *Proceedings of the Second International Workshop on Statistical and Computational Theories of Vision*, 2001.
- [32] D. Murray and A. Basu, "Motion Tracking with an Active Camera", *IEEE Trans. on Pattern Analysis and Machine Intelligence*," vol. 16, no.5, pp. 449-459, 1994.
- [33] M. Isard and A. Blake, "Condensation – conditional density propagation for visual tracking", *International Journal on Computer Vision*, 29(1), 1998, pp. 5-28.
- [34] Y. Meng, A Mobile Vision System with Reconfigurable Intelligent Agents, *2006 IEEE World Congress on Computational Intelligence (WCCI 2006)*, July 16-21, 2006, Vancouver, BC, Canada.