

Agent-based reconfigurable architecture for real-time object tracking

Yan Meng

Received: 16 November 2007 / Accepted: 19 February 2009 / Published online: 17 March 2009
© Springer-Verlag 2009

Abstract Most conventional object tracking algorithms are implemented on general-purpose processors in software due to its great flexibility. However, the real-time performance is hard to achieve due to the inherent characteristics of the sequential processing of these processors. To tackle this issue, a reconfigurable system-on-chip (rSoC) platform with microprocessors and FPGAs is applied in this paper. To simplify the hardware/software interface, a Belief–Desire–Intention (BDI)-based multi-agent architecture is proposed as the unified framework. Then an agent-based task graph and two heuristic partitioning methods are proposed to partition the hardware and software on an rSoC platform. Compared to the module-based architecture, this BDI-based multi-agent architecture provides more efficiency, flexibility, autonomy, and scalability for the real-time tracking systems. A particle swarm optimization (PSO)-based object detection and tracking algorithm is applied to evaluate the proposed architecture. Extensive experimental results of object tracking demonstrate that the proposed architecture is efficient and highly robust with real-time performance.

Keywords Object detection and tracking · Hardware/software partitioning · BDI agent · Reconfigurable system-on-chip · Task graph

1 Introduction

Real-time object tracking is an active research area, which has attracted extensive attentions from multi-disciplinary fields, and it has various applications, such as service robots, medical care robots, surveillance systems, and public security systems. Real-time performance is critical for object tracking under dynamic environment. However, most available computer vision approaches for object tracking are complex and computational expensive with some underlying assumptions, such as large memory, high computation power, static vision system, or off-line processing. In a real-world object tracking system, the system has limited processing resources and memory, and it has to be highly robust and adaptive to changing environments with real-time constraints, where lightening conditions may change, occlusions may happen, or object may move out of focus of view of cameras.

Conventionally, the complex tracking algorithms are implemented on general purpose processors 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. 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. A software-only solution would push the limit of the processing capability, which may lead to a very conservative computation solution and a very challenging real-time scheduler.

On the other hand, the specific hardware can respond to external inputs more efficiently since 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

Y. Meng (✉)
Department of Electrical and Computer Engineering,
Stevens Institute of Technology,
Hoboken, NJ 07030, USA
e-mail: yan.meng@stevens.edu

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 computing devices. The newest FPGA technology allows a designer to use a single reconfigurable platform to instantiate both processors and required logic units, which directly leads to feasibility of the reconfigurable system-on-chip (rSoC) architecture. This rSoC platform is desirable for real-time tracking systems, which must be highly responsive to dynamic environments.

Based on this rSoC architecture, a Belief–Desire–Intention (BDI) agent model is proposed in this paper 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 decomposed into agents based on system specifications, where these agents can accomplish some specific tasks independently and can communicate with each other.

This rSoC architecture raises the possibility of different HW/SW partitioning approaches. In the design of systems with heterogeneous computing resources, such as microprocessors and FPGAs, one of the critical steps is the allocation of the computation of an application onto different computing components. This system partitioning problem plays a dominant role in the system cost and performance. An agent-based task graph is proposed as the system partitioning tool, and two heuristic methods, centralized and decentralized methods, are presented to partition agents into different computing resources. Compared to the regular module-based architecture, the BDI-based multi-agent model with the heuristic HW/SW partitioning would provide more efficiency, flexibility, autonomy, and scalability.

Finally, to evaluate the proposed system architecture and system partitioning methods, a particle swarm optimization (PSO)-based object detection and tracking algorithm proposed in our previous work has been implemented on this rSoC platform for a real-time object tracking system under various scenarios.

The paper is organized as follows. Section 2 describes some related work. A BDI-based multi-agent architecture is proposed in Sect. 3. Section 4 presents system partitioning where the agent-based task graph and two heuristic HW/SW partitioning methods are proposed. Section 5 presents a real-time tracking system using a PSO-based

method. Experimental results on a real-time tracking system are discussed in Sect. 6. The paper is concluded in Sect. 7 with the future work.

2 Related work

Recently, the agents start to be considered to have great potential to be used in software engineering techniques. Agent-Oriented Software Engineering (AOSE) was proposed by Jennings and Wooldridge [11], which opened up some related research areas for distributed methodologies to model real-time complex systems. Later, more research work have been conducted to apply multi-agent architecture to real-time systems. Micacchi and Cohen [18] proposed a multi-agent architecture for robotic systems in real-time environments with software-only platform. They mainly focused on how to add more autonomy to agents for new tasks in demand. Carrascosa et al. [4] proposed a hybrid multi-agent architecture, called SIMBA, for real-time tasks. Basically, Case-Based Planning BDI deliberative agents collaborate with ARTIS agents to solve real-time problems. Here, an ARTIS agent is an agent that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors. The BDI agent model for multi-agent systems was proposed by Kinny et al. in [14]. Wood [31] proposed a MaSE architecture approach for multi-agent systems. Brazier et al. [3] proposed a high-level modeling framework, called DESIRE (Design and Specification of Interacting Reasoning components), for multi-agent systems. An IDEF approach-based methodology was proposed by Kendall et al. for the engineering of agent-based systems in [13].

However, all of these approaches have been designed and developed only for the engineering of software agents. For complex real-time systems, sometimes it is hard to achieve the desired real-time performance using software-only platforms. On the other hand, the recent advance 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 [15, 16]. In our previous work [16], a BDI agent architecture was proposed on a FPGA-based platform for a mobile robot navigation system, where several dynamic reconfiguration modules were proposed for the FPGA-based hardware agents. Fleischmann et al. [9] 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. Context switching FPGA architectures were also proposed by Scalera and Vazques [26] and Sckanina and Ruzicka [27], 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.

As the soft processor cores for FPGAs become available in the rSoC platform, the HW/SW partitioning issues becomes critical in the overall system design procedure. Some of the more recent HW/SW co-design research used reconfigurable processors as the platform for implementation [2, 15], where their architectures combined a reconfigurable functional unit with the microprocessor. Partitioning is a well-known NP-hard problem and many heuristics have been proposed to guide the partitioning of a system into hardware and software components [19, 25]. Saha et al. [25] modeled the HW/SW partitioning problem as a Constraint Satisfaction Problem (CSP), and a genetic algorithm-based approach was proposed to solve the CSP to obtain the partitioning solution. Osterling et al. [20] and Wianton et al. [30] have conducted system partitioning on the functional/task level. A runtime partitioning system, Dynamo, for FPGA-based HW/SW image processing systems was proposed by Quinn et al. [21], which contained a mix of processors and reconfigurable hardware. Dynamo could dynamically select the most efficient combination of HW and SW component implementations to minimize pipeline runtime and generate the source code that implements the pipeline. Walker et al. [29] proposed a holonic, multi-agent systems approach for manufacturing job shop scheduling, where instead of developing some heuristic schedulers directly, their used an evolutionary algorithm to evolve the mixed heuristics scheduler.

3 A BDI-based multi-agent architecture

First, we will introduce the BDI agent in this section, and then propose the BDI agent-based task graph in next section. An *agent* is an independent processing entity that interacts with the external environment and with 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 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 environment. The input/output port is used to send and receive information to and from the environment. The inter-agent communication port allows the agents to send/receive information to/from other agents and cooperate with others.

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 follows:

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

This structure can be implemented by high-level software languages on microprocessors, or Verilog HDL on FPGA. This is a very general definition of the BDI agent. To make it easy to understand, a detailed definition of a particle agent is provided in Sect. 5 for a real-time object tracking method.

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 user.
- *Social ability* To effectively change or interrogate their environment, BDI agents possess the ability to communicate with the outside world through their input/

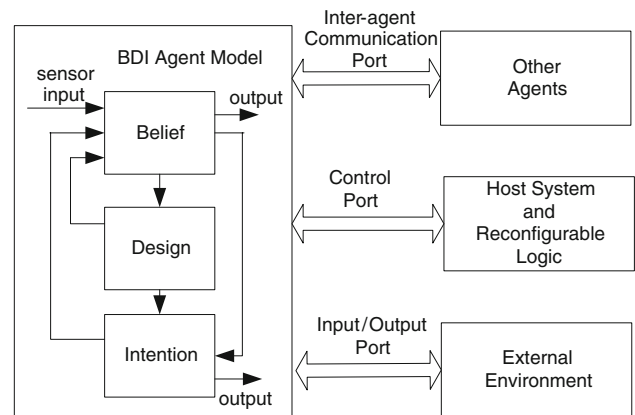


Fig. 1 The BDI agent model

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 affect actions that achieve their goals by taking the initiative.

Some agent-oriented implementations have been developed, such as JACK [1] and distributed multi-agent reasoning system (DMARS) [6]. In our system, 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 environment 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 the relevant plans are exhausted. Usually, a fitness function is used to evaluate the performance of the current plan. If the fitness value of the current plan is greater/less than the predefined threshold depending on how to define the fitness function, then we can say that the current plan fails.

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 System partitioning

4.1 Agent-based task graph

In the design of systems with heterogeneous computing resources, one of the critical steps is the allocation of the computation of an application onto the different computing components. This system partitioning problem plays a dominant role in the system cost and performance. In this paper, we will focus on the system partitioning on the task/function level.

To tackle this system partitioning problem, we propose a BDI agent-based task graph, where the tasks can be represented by the BDI-based agent model. The agent-based task graph is a directed acyclic graph, which describes the precedence relationship between the agents. The advantages of using a BDI-based agent model to define a task are

listed in the followings. First, all the tasks can be defined as the same standard BDI model, which has the same interfaces between the agents and the system, and between the agents and other agents with precedence constraints. One agent can consist of one or multiple tasks depending on the applications. This feature simplifies the system synchronization as well as HW/SW interfaces. Second, it is possible to add more intelligent features, such as autonomy, social ability, reactivity, and proactivity, to the tasks represented by BDI agents. These intelligent features could provide an advanced general platform for various partitioning and scheduling algorithms.

Figure 2 shows an example of an agent-based task graph with two different resources: PowerPC microprocessor and FPGA. An agent-based task graph is defined as $TG = (A, L)$, where A is a set of agent nodes, $A = \{a_0, a_1, a_2, \dots, a_n\}$, which define different functional tasks in the application and L is a set of links connecting the agents. As shown in Fig. 2, an agent-based task graph has two special nodes a_0 and a_n , which are virtual nodes to ensure a unique starting and ending point of the task graph. A link $l_{ij} \in L$ represents an immediate precedence constraint between agent a_i and a_j . Basically, the agents at different precedence levels should be executed sequentially from the top to the bottom, while the agents at the same precedence level should be executed in parallel and can be partitioned into different computing components. In the task graph, an agent can only be executed when all the agents with higher precedence level have been executed.

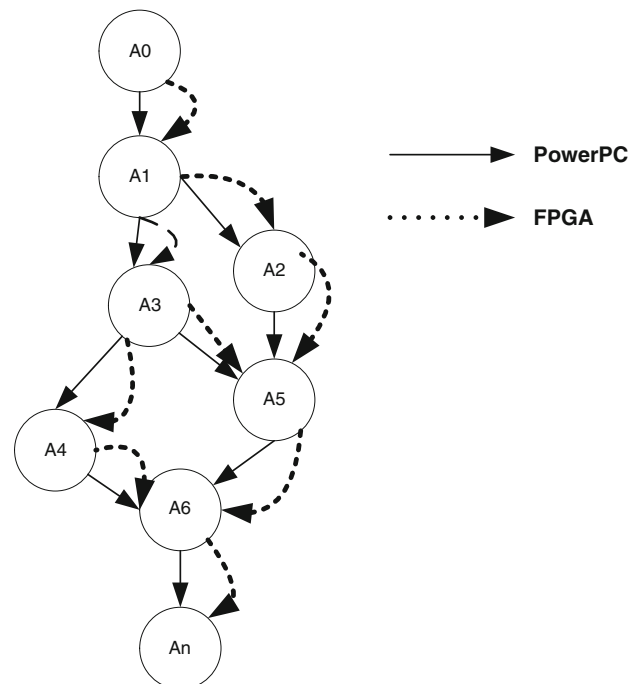


Fig. 2 The agent-based task graph

If there are r different computing resources and N agents in a given real-time system, then there will be up to r^N independent partitioning solutions. Therefore, considering the dependency constraints between the agents, the system partitioning problem can be defined as: given r different computing resources and N agents in a real-time system, we need to find an optimal partition solution $P = \{P_1, P_2, \dots, P_r\}$ which can satisfy the precedence constraints on r computing resources, where P_i represents the agents that are assigned to computing resource i , $P_i \cap P_j = \phi$ so that the overall execution time can be minimized.

4.2 Heuristic partitioning rules and constraints

In general, the following partitioning policies are proposed for the initial partitioning.

1. 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.
2. 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.
3. The agents with hard deadlines are distributed to hardware, while the agents with soft deadlines may be implemented in software.
4. The hardware agents can be partitioned concurrently if no precedence constraint exists.

Based on the above general partitioning rules, different rSoC platforms would also provide different constraints for the system partitioning. In this paper, Xilinx ML310 embedded programmable system is applied as our target platform to evaluate the proposed architecture and partitioning algorithms.

ML310 platform, as shown in Fig. 3, offers designers a Virtex-II Pro XC2VP30-based embedded platform. It provides 30 k logic cells, over 2,400 kb BRAM, and dual PowerPC 405 processors, as well as onboard Ethernet MAC/PHY, 256 M 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.



Fig. 3 The ML310 embedded platform

An on-demand message passing (ODMP) communication protocol proposed in our previous work [17] is applied for agent communication. This communication mechanism provides transport independence, and therefore, it is portable to different agent-based real-time systems without significant modification to both software and hardware. The only parts that need to be customized are adding different transport-dependant adapters to new systems.

The underlying physical mechanism for the multi-agent communication is through shared memory. 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.

This platform imposes several constraints on the partitioning problem. The code length of the PowerPC processor must be less than the size of the instruction memory, and the agents implemented on FPGAs must not occupy more than the total number of available CLBs. The execution time and required resources for each agent on different recourses depend on the implementation of the agent. It is assumed that the tasks corresponding to the agents are static and pre-computed. Some constraints come from the FPGA platform itself. For example, in Virtex-II pro FPGA, the dynamic reconfiguration can only be executed on column base. Therefore, even if there are still some CLBs available under one unfinished column, we cannot partition the new agent onto those CLBs.

Agents allocated on the PowerPC processor are executed sequentially subject to the precedence constraints

within agents. Concurrent agents may execute in parallel on different system resources. The system constraints are used to determine whether a particular partition solution is feasible. For all the feasible partitions that do not exceed the capacity constraints, the partitions with the shortest execution time are considered the best.

Estimation is carried out for each agent node to get performance characteristics, such as execution time, software code length, and hardware area. Based on the specification data of ML310 manual, we can get the performance characteristics for each type of operations. Using these operation (instruction) characteristics, the performance of each basic block can be estimated. This information for each agent node is used to evaluate a partitioning solution. In addition, the hardware cost and software code length for each agent are used for the estimation of solution. The software code length is estimated based on the number of instructions needed to encode the operations. The hardware cost is estimated by the approximate area needed to implement the agent on the reconfigurable logic.

4.3 Off-line centralized partitioning method

Since the objective of this partitioning is to achieve the optimal execution time subject to hardware area constraint, therefore, the fitness function is defined as the function of execution time and hardware area:

$$f = \sum_{i=1}^N \sum_{j=1}^r f_{ij} = \sum_{i=1}^N \sum_{j=1}^r w_1 t_{ij} + w_2 h_{ij}, \quad (1)$$

where t_{ij} and h_{ij} are the execution time and hardware area used for allocating agent i to resource j , respectively, N is the total number of agents, r is the total number of computing resources, and w_1 and w_2 are the weight factors. Here, only one PowerPC processor and FPGAs are used as two computing resources.

Based on this fitness function, a centralized partitioning algorithm is proposed first. It is assumed that there is a central control system which conducts the partitioning based on the parameters of all agents as well as the system constraints from a global point of view. The centralized partitioning algorithm is summarized as follows:

1. Initialization: generate BDI-based agent nodes on the task graph, and define the precedence constraints between the agents.
2. Allocate all of the agents to SW and HW using the general partitioning rules listed in Sect. 4.2.
3. Evaluate the partition using the fitness function (1). If the fitness value is less than a predefined threshold, go to step 5. Otherwise, go to step 4. Here, the predefined

threshold is problem dependent. In other words, it only depends on the real-time performance requirement of the interested problem.

4. If the execution time is greater than the real-time requirement and there is still some HW area available, move the agents with longer execution times from SW to HW until the usage of HW is maximized. If the execution time is greater than the real-time requirement and there is no more HW area available, try to swap the SW agents with longer execution times with those HW agents with shorter execution times.
5. Stop the iteration if the maximum number of iterations has been reached or the overall fitness value is less than a predefined threshold. Otherwise, go to step 3.

4.4 Online decentralized partitioning method

The centralized method is usually efficient for off-line partitioning in most cases. However, in most real-time systems, it is very hard, if not impossible, to predict all the tasks or events before runtime, which makes the off-line methods difficult to be applied because it is hard to add the unpredictable and uncertainty into the off-line method. It is desirable to propose a method which can dynamically partition the agents into different computing resources based on the task parameters of each agent, such as execution time, release time, deadline, and dependency. However, the dynamic partitioning is very complex if it has to handle some sporadic tasks due to environmental changes.

To address this issue, a decentralized method is proposed, where each agent makes its own decisions to select the computing resources based on their own parameters as well as the shared information among the agents from a global memory. The global information includes the current partitioning status, the availability of computing resources, and the execution time for each agent. The agents who have precedence constraints would communicate with each other to share the information through the shared memory mechanism, where ODMP [17] communication protocol is applied to pass messages. Each agent has to make its own partition decisions in a distributed manner. In this manner, whenever there is an unpredictable change, the agent can always make decisions to dynamically adapt to the change on the fly instead of reschedule everything from scratch like the centralized method.

The fitness function for each individual agent is defined as follows:

$$f_{ij} = w_1 t_{ij} + w_2 h_{ij}, \quad (2)$$

where t_{ij} and h_{ij} are the execution time and hardware area used for allocating agent i to resource j , respectively, w_1

and w_2 are the weight factors to adjust the balance between the execution time and hardware area.

This method is summarized as follows:

1. Initialization: generate BDI-based agent nodes on the task graph, and define the precedence constraints between the agents.
2. Start partitioning the top agents in the task graph to the bottom ones. Each agent makes its own decision to pick the computing resource (HW or SW) using the general partitioning rules listed in Sect. 4.2.
3. By checking the shared memory, the overall fitness value is calculated by summing up all the fitness value of each agent. If the overall fitness value is less than a predefined threshold, go to step 5. Otherwise, go to step 4. Here, the predefined threshold is problem dependent. In other words, it only depends on the real-time performance requirement of the interested problem
4. At each iteration, two cases are considered:
 - a. If there are some HW areas available, each SW agent would do the following actions sequentially until all SW agents have finished the following actions or no more HW area is available.
 - i. Calculate the difference between the predefined threshold and the actual fitness value.
 - ii. Based on its own current execution time and hardware usage, this agent would move to HW if the estimated reduced execution time is greater than a predefined threshold.
 - iii. Update the global information if any change is made.
The fitness function for each agent is defined as (2). Go to step 5.
 - b. If no more HW area is available, SW agents have to decide to swap with HW agents to reduce the execution time.
5. Stop the iteration if the maximum iteration has been reached or the overall fitness value is less than a predefined threshold. Otherwise, go to step 3.

By using this decentralized automatic system partitioning method, some unique features of the BDI-based agent architecture can be achieved, such as autonomy (making its own decision based on its environment and current status), social ability (communicate with other agents to share information), reactivity (adapt to new changes if necessary), and proactivity (initiating new actions if necessary to achieve optimal goal). Once the environment has changed, some agents may have to be reallocated to different resources to adapt to this change.

4.5 Dynamic reconfiguration

During the online partitioning procedure, dynamic reconfiguration is necessary to implement on the FPGA platform. Dynamic reconfiguration is based on the idea that parts of the system remain available during the reconfiguration. The reconfiguration module should introduce minimal overhead during normal operation. Deploying dynamic run-time reconfiguration in systems results in reduced chip area and power consumption. 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.

A self-reconfiguration module proposed in our previous work [17] is utilized here for hardware dynamic reconfiguration. Since this is not the focus of this paper, we only provide the general idea of this self-reconfiguration framework here. For the detailed description of the self-reconfiguration module, please refer to [17].

The aim 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. Since the Vertex Pro II FPGA is chosen as the RMCSoc platform in our system, the internal configuration access port (ICAP) in Vertex Pro II FPGA makes the self-reconfiguration possible. Generally, the logic circuits are divided into fixed logic part and reconfigurable logic part. 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.

In summary, this self-reconfiguration platform provides two advantages. First, it provides an autonomous method

for the hardware agents to be reconfigured due to dynamic environment under real-time constraints. Second, it provides a feasible physical mechanism to allow the ODMF communication protocol to be implemented on this platform using shared memory features.

5 Real-time object tracking system

Some research have been conducted for people detection systems using one single cue at the detection, such as motion [23], color [22], contour [5], or face's features (such as eyes, nose, mouth, etc.) [24]. In a natural, complex, and dynamic working environment, the 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 [8, 10, 28].

To evaluate the proposed agent-based architecture, in this paper, we implement a PSO-based object tracking algorithm proposed in our previous work [32] in this rSOC platform, which combines the color and motion cues together. The PSO-based object tracking method is summarized here.

Usually, a rectangle window is used to identify the interested object in an image. Four parameters can describe a rectangle window, including a 2D location of the central point represented by x and y , length l , and width w , as shown in Fig. 4. Instead of searching for object features in the image itself, we map these feature parameters to a virtual point in a four-dimensional search space, and propose a PSO-based searching method to search in this high-dimensional virtual space.

The PSO algorithm is an efficient optimization method proposed by Kennedy and Eberhart [7] from the simulation of a simplified social model, which obviously has its root in artificial life in general, and in bird flocking, fish schooling and swarming theory in particular. In the PSO algorithm each individual is called a "particle," and is subject to a movement in a multidimensional space that represents the

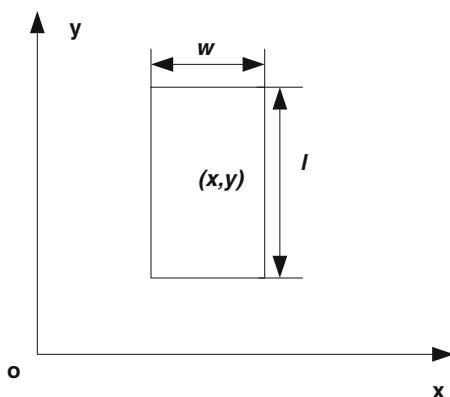


Fig. 4 Four parameters associated with a particle window

belief space. Particles have memories, thus retaining part of their previous states. Each particle's movement is the composition of an initial random velocity and two randomly weighted influences: individuality, the tendency to return to the particle's best previous position, and sociality, the tendency to move toward the neighborhood's best previous position. The goodness of a position is defined by the fitness function, whose format depends on explicit applications.

The velocity and position of the particle at any iteration is updated based on the following equations:

$$v_{id}^{t+1} = w \cdot v_{id}^t + c_1 \cdot r_1 \cdot (p_{id}^t - x_{id}^t) + c_2 \cdot r_2 \cdot (p_{gd}^t - x_{id}^t), \quad (3)$$

$$x_{id}^{t+1} = x_{id}^t + v_{id}^{t+1}, \quad (4)$$

where v_{id}^t is the component in dimension d of the i th particle velocity in iteration t , x_{id}^t is the component in dimension d of the i th particle position in iteration t , c_1 , c_2 are constant weight factors, p_{id}^t is the best position achieved by particle i , p_{gd}^t is the best position found by the neighbors of particle i , r_1 , r_2 are random factors in the $[0,1]$ interval, and w is the inertia weight. The PSO requires tuning of some parameters: the individual and sociality weights c_1 , c_2 , and the inertia factor w . According to (3), each particle adjusts its velocity by combining three forces: keeping the velocity of last moment, moving to the best position from its own memory, moving to the best position found by its neighbors.

In the case of rectangle window searching, each particle represents a search window with specific values of parameters, which can be defined as:

$$P = \{p_i | p_i(x_i, y_i, l_i, w_i), \quad i = 1, 2, \dots, N\}, \quad (5)$$

where x_i and y_i represent the central point of the rectangle related to particle i ; l_i and w_i represent the length and width related to particle i , respectively; and N is the population of the swarm particles. Each individual particle has different values of these parameters. In other words, they are distributed in a four-dimensional search space. Then a PSO-based algorithm is applied to search for one or multiple best virtual points which represent the optimized or near optimized parameters for the associated tracking windows.

First, the image is transformed from RGB format into HSV format. The values of hue are abstracted to build a histogram for each window. The fitness function of each particle is defined as the intersection between this histogram and the object histogram:

$$\begin{aligned} f_i(x_i, y_i, l_i, w_i) &= H(h_i(x_i, y_i, l_i, w_i), g(x_r, y_r, l_r, w_r)) \\ &= \text{norm} \left(\sum_{j=1}^{256} \frac{\min(h_j, g_j)}{\max(h_j, g_j)} \right), \quad i = 1, 2, \dots, N. \end{aligned} \quad (6)$$

where $h_i(x_i, y_i, l_i, w_i)$ and $g(x_o, y_o, l_o, w_o)$ represent the histograms of search window i and the object model,

respectively. $H(h_i, g)$ is the histogram intersection between the h_i and g , which is the sum of overlaps along all possible pixel values. Since the tracking window may vary compared to the object model due to object movement, the normalized value of histogram intersection is calculated to eliminate the size effect. The normalized value is between 0 and 1. The higher the histogram intersection, the more similar the two histograms are.

During object detection phase, a large number of particles need to be generated to search for the most matched points within the searching space. Once the object is detected, the tracking algorithm takes over. The tracking phase can be expedited with the motion constraints of object assuming that the most possible location of new tracking window should be close to the position of last tracking window. Therefore, smaller number of particles would be good enough since the searching area has been limited to a certain area. To accelerate the processing, the detection algorithm can be skipped under normal tracking condition.

Usually, detection phase takes longer than tracking phase because the searching area of the detection phase is much larger than the tracking phase. The searching area for the detection phase is the whole image, while the searching area of the tracking phase is just a small portion of the image.

Initially, tracking system has a referenced object histogram feature and continuously runs object detection algorithms using PSO-based method. Only those windows whose fitness values exceed a predefined threshold can be selected. Then the selected feature objects are passed to tracking algorithm to track their motions in consequent images using the object motion constraints. Only the best matched one in the tracking part will be treated as the object.

Under dynamic environment, the light condition, lens focus, object location, occlusions, or other parameters may change, the histogram of the predefined object model would have to adapt to the dynamic situations. In the case where the light condition is changed gradually, the new object model can be estimated by integrating the original object model with the best fit window in the previous frame using the following equation:

$$g_{\text{new}}(x_r, y_r, l_r, w_r) = k_1 g_{\text{old}}(x_r, y_r, l_r, w_r) + k_2 h_j(x_j, y_j, l_j, w_j), \quad (7)$$

where k_1 and k_2 are the weight factors and h_j is the best fit window in the previous frame.

If the tracked object is occluded partially by other objects, the proposed PSO-based tracking window would eventually converge to the non-occluded part of the object since the histogram value is independent of the window

size. For the same reason, when the occlusion is released, the object can be easily recovered to its original window size.

To demonstrate how to apply the proposed BDI architecture to the specific SW/HW agent, we use a particle agent as an example here. A particle agent can be defined as follows:

```
<Particle Agent>
{
  <Beliefs>
  {current state; neighborhood information; local
  memory for previous state and best local state in
  the history;}
  <Desires>
  {if the fitness value calculated from equation (6) is
  less than a predefined threshold, stop the agent
  procedure; otherwise, continue.};
  <Intentions>
  update the current state using equation (3) and (4);
  update the neighborhood information; update the
  local memory.}
}
```

6 Experimental results

To evaluate the system performance of the proposed agent-based architecture on rSOC platform, the PSO-based object detection and tracking algorithm is implemented as our benchmark on the ML310 platform. Fifteen particles fly in 320×240 images, where only five iterations are needed to achieve the convergence. In software, we define the particle as one class with one unified data structure, where each particle is defined as one instance of this class and has different values of input/output variables. In FPGA, all particles have the same logic connections, where each particle has different values of input/output variables.

There are six candidates of on-chip memory network topologies described by Kearney and Veldman [12]. The star network configuration is used to implement the on-chip network for use for its ease of implementation, support of concurrency, and low latency.

To evaluate the proposed approach, a tracking video experiment is conducted first, as shown in Fig. 5. Figure 5 shows a tracking process containing occlusion. A student walks in an office environment, hides himself behind a box, and reappears from the box. This procedure includes occlusion, disappear, and reappear situation. The proposed algorithm showed the robustness to keep tracking the object all over the cases.

For each case, 35 runs have been conducted for the centralized and decentralized partitioning methods. To provide



Fig. 5 A tracking experiments of a single person with full occlusion

the comparison, the random sampling method and software-only solution have also been conducted. The terminal condition of the iteration is set up as 30 times. In the experiments, we set $w = 1.0$, $c_1 = c_2 = 2.0$, $k_1 = 0.3$, $k_2 = 0.7$, $w_1 = 0.35$, and $w_2 = 0.65$. The agent size for the task graph in the implementation is 15 since the implementation using 15 agents can provide us quick convergence based on our experiments.

Figures 6 and 7 show the execution time comparison, mean value, and variance, between the proposed heuristics centralized partitioning method, decentralized partitioning method, random sampling, and software-only solution. The task graph size is 15 agents for all the test cases.

Here, the execution time is the time it takes for processing a single frame. It is obviously that the software-only solution takes much longer time to finish the tracking process in a single frame, while the centralized partitioning method and decentralized partitioning method have similar performance and both outperform the randomly sampling method. The underlying reason for this improvement is that all particle agents can implement their searching behaviors in parallel in hardware solution instead of sequential execution in software solution. With the heuristic partitioning methods, the system can process about 30 frames per second, which provides a real-time performance for the tracking task.

Table 1 summarizes the average breakdown of hardware resources of ML310 platform used by hardware agents for centralized and decentralized partitioning methods. It can

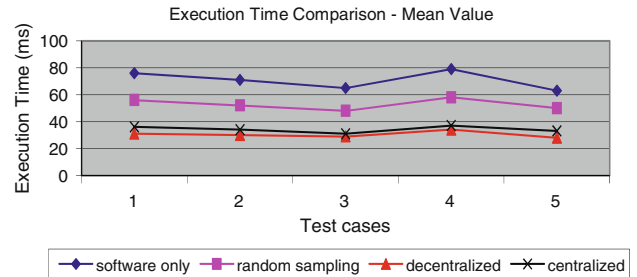


Fig. 6 The mean value of the execution time comparison

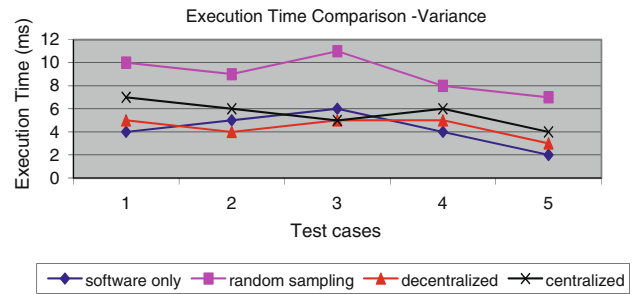


Fig. 7 The variance of the execution time comparison

Table 1 The breakdown of hardware resources used by hardware agents

Resources	Centralized (%)	Decentralized (%)
block RAMs	78	74
CLBs	79	75
Function generators	81	77

be seen that most of the hardware resources have been utilized.

To compare the performance of the proposed PSO-based tracking method with other object tracking methods, a general particle filter (PF) method and a general mean-shift method have also been implemented. As shown in Fig. 8, there are two people in an indoor environment, where one people’s face is defined as the tracking object. As seen from Fig. 8(a1) and (b1), the tracked object is getting closer and starts to be occluded by another people’s face. When the object reappears, it attracts the tracking window back as shown in Fig. 8(c1). The object tracking is recovered and continued in Fig. 8(d1). Here, the proposed PSO-based algorithm shows its higher robustness under dynamic occlusions.

Compared with the general PF method, as shown in Fig. 8(a2) and (b2), both PSO-based method and the general PF method are capable of tracking the object without occlusion. However, when the occlusion occurs, the general PF is attracted by the cluttered background and cannot recover from the occlusion, as shown in

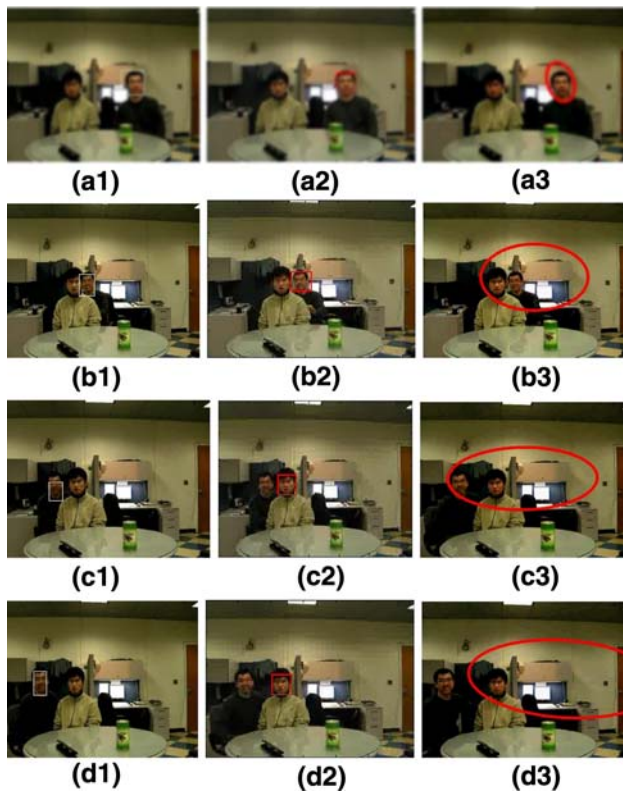


Fig. 8 Indoor tracking experiments of two persons with dynamic occlusion using a PSO-based method, a general PF method, and a general mean-shift method. First column (a1)(b1)(c1)(d1) shows the results of the proposed PSO-based method, the second column (a2)(b2)(c2)(d2) for a general PF method, and the last column (a3)(b3)(c3)(d3) for a general mean-shift method. From (a1) to (d1), the PSO-based method is capable of tracking the object with short occlusion. However the general PF lost the object when the object is occluded by cluttered backgrounds, as shown in (a2) and (d2). The mean-shift method lost the object at the very early stage and keeps drifting away in the whole process

Fig. 8(a2)–(d2). When the face in track is occluded by another face, the tracking window using the general PF method drifts to another face, instead of the original tracking face. This is reasonable because another face has similar appearance with the face in track. However, when the face in track reappears, the general PF cannot explore widely enough to catch up the reappeared tracking object. Therefore, it still sticks to the other face and loses the interested object. On the other side, the PSO-based algorithm is robust with occlusion and capable of re-catching the object, as shown in Fig. 8(c1) and (d1). The reason for this robustness is because when the object reappears, the particles start searching around in a wider area with different window sizes and locations and can easily detect the tracked object. Since the face in track has the highest fitness value compared to another face, it will attract more particles, which leads to the object being tracked correctly.

PSO-based method is also compared with the mean-shift method. The ‘camshift’ program from OpenCV is adopted here as the mean-shift program. The proposed PSO-based algorithm keeps tracking the object from Fig. 8(a1)–(d1), while the mean-shift method gradually shifts off from the object and loses it eventually as shown from Fig. 8(a3)–(d3). The exploring capability of particles supplies more flexible adjustment in a high-dimensional space, which leads to more accurate size of the tracking window.

In the experiments, the PSO-based method demonstrates its adaptive capability and robustness for the object tracking under dynamic indoor. This robustness comes from the flexibility and the sharing mechanism of the PSO algorithm, which allows individual agents to explore new areas as well as to learn from the neighbors. The initial wide distribution of particles naturally makes this algorithm to handle image jitters between image frames. Finally, the proposed PSO-based method is easy and quick to implement to achieve real-time performance.

To verify the robustness and efficiency of the proposed online decentralized partitioning method compared with the off-line centralized method, we implement the PSO-based object tracking algorithm on the ML310 FPGA platform using the BDI agent-based architecture for three cases. Table 2 gives the recovery execution time comparison.

The first case is a car tracking experiment with a small occlusion by the text in the middle of the image in an outdoor environment, as shown in Fig. 9. When the car is moving away, the tracking window becomes smaller but still tracks the car. Second case is the single person tracking in an indoor environment with big occlusion and even disappearance, as shown in Fig. 5. Third case is a people tracking crossing over with another people in an indoor environment, as shown in Fig. 8.

As seen in Table 2, the results show that the decentralized method outperforms the centralized method in the recovery time for all three cases. The recovery time of the first case is smaller compared to other two cases because the occlusion in the first case is very small and would not cause the system to switch from the tracking mode to the detection mode. Since the system cannot predict the dynamic environment, when the object is completely occluded, which happens in the last two cases, the system is still under tracking mode. The tracking algorithm is

Table 2 The recovery time of the proposed PSO-based algorithm with different cases

Cases	Centralized (ms)	Decentralized (ms)
Outdoor car tracking	26.76	25.23
Indoor single person tracking	35.84	31.58
Indoor two people crossover	34.19	29.64



Fig. 9 An outdoor environment for a car tracking

carried out near the tracking window of previous frame and none of the particle window is selected. Then the system has to switch to the detection mode, and the detection algorithm is conducted on the entire image. This mode-switching procedure imposes additional latency into the system. Therefore, compared to the off-line centralized method, the online decentralized method is more robust, flexible, and can handle unpredicted and unexpected events during runtime, which is very critical for most real-time systems.

7 Conclusion and future work

This paper has proposed a BDI agent-based architecture for a reconfigurable embedded platform for a real-time tracking system. This agent-based architecture establishes a unified framework, which significantly simplifies the HW/SW partitioning and communication. Furthermore, on top of this BDI-based multi-agent architecture, two heuristic partitioning methods are proposed, centralized, and decentralized methods, aiming at improving the overall real-time performance of tracking system. Compared to the traditional module-based architecture, more unique features, such as autonomy, social ability, reactivity, and proactivity, can be embedded into the systems with BDI agent-based architecture, which provides a very powerful and general platform for distributed dynamic partitioning for various real-time systems.

The proposed BDI-based multi-agent architecture provides a very flexible platform for the future extensions, such as learning capability and proactivity to adapt to the dynamic environments. Our future research will focus on dynamic system partitioning and scheduling for real-time detection and tracking systems. 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 environmental changes. For example, if an agent can memorize previous partitioning patterns with good performance based on a specific environment, when a similar situation happens again, the agent would use the previous patterns from its own memory instead of trying to relocate the resources from scratch. If all the previous patterns cannot satisfy the new requirements, then a new pattern will be generated and the agent's memory will be updated for future references. Overtime, the system would learn to adapt to environmental changes very quickly for real-time performance. Furthermore, the decentralized partitioning method can be easily extended to distributed real-time embedded systems, which will be investigated in the future as well.

References

1. Agent-Oriented Software Pty Ltd: JACK Manual (v4.1). <http://www.agent-oriented.com>, (2003)
2. Baleani, M., Gennari, F., Jiang, Y., Patel, Y., Brayton, R.K., Sangiovanni-Vincentelli, A.: 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 (2002)
3. Brazier, F., Keplicz, B.D., Jennings, N., Treur, J.: Desire: modelling multi-agent systems in a compositional formal framework. *Int. J. Cooperative Inf. Syst.* **6**(1), 67–94 (1997)
4. Carrascosa, C., Bajo, J., Julian, V., Corchado, J.M., Botti, V.: Hybrid multi-agent architecture as a real-time problem solving model. *Expert. Syst. Appl.* **34**(1), 2–17 (2008)
5. Davis, J., Sharma, V.: Robust detection of people in thermal imaging. In: 17th International Conference on Pattern Recognition (ICPR'04), 713–716 (2004)
6. d'Inverno, M., Kinny, D., Luck, M., Wooldridge, M.: A formal specification of dMARS. In: Proceedings of the 4th International Workshop on Agent Theories, Architecture, and Language, vol. 1365 of LNAI, pp. 155–176 (1998)
7. Eberhart, R., Kennedy, J.: A new optimizer using particles swarm theory. In: Proc. Sixth International Symposium on Micro Machine and Human Science, IEEE Service Center, Piscataway, NJ (1995)
8. Feyrer, S., Zell, A.: Detection, tracking, and pursuit of humans with an autonomous mobile robot. In: Proceedings of the International Conference on Intelligent Robots and Systems (IROS'99), pp. 864–869 (1999)
9. Fleischmann, J., Buchenrieder, K., Kress, R.: Codesign of embedded systems based on java and reconfigurable hardware components. In: Design Automation and Test in Europe (1999)
10. Froba, B., Kublbeck, C.: Face detection and tracking using edge orientation information, *SPIE Vis. Commun. Image Process.* **4310**, 583–594 (2001)
11. Jennings, N.R., Wooldridge, M.: Agent-oriented software engineering. Handbook of Agent Technology. In: Bradshaw, J. (ed.) AAAI/MIT Press, Cambridge (2000)
12. Kearney, D., Veldman, G.: A concurrent multi-bank memory arbiter for dynamic IP cores using idle skip round robin. In: Proceedings of IEEE International Conference on Field-Programmable Technology (FPT '03), pp. 411–414, Tokyo, Japan (2003)

13. Kendall, E.A., Malkoun, M.T., Jiang, C.H.: A methodology for developing agent based systems. In: Proceedings of the First Australian Workshop on DAI: Distributed Artificial Intelligence: Architecture and Modeling, Lecture notes in Computer Science, vol. 1087, pp. 85–99 (1995)
14. Kinny, D., Georgeff, M., Rao, A.: A methodology and modeling technique for systems of BDI agents. In: Proceedings of the 7th European workshop on modeling autonomous agents in a multi-agent world, pp. 56–71 (1996)
15. Li, Y., Callahan, T., Darnell, E., Harr, R., Kurkure, U., Stockwood, J.: Hardware–software co-design of embedded reconfigurable architectures. In: Design Automation Conference (2000)
16. Meng, Y.: A dynamic self-reconfiguration mobile Robot navigation system. In: IEEE/ASME International Conference on Advanced Intelligent Mechatronics, pp. 1541–1546 (2005)
17. Meng, Y.: An agent-based reconfigurable system-on-chip architecture for real-time systems, In: 2nd International Conference on Embedded Software and Systems, Xian, China (2005)
18. Micacchi, C., Cohen, R.: A multi-agent architecture for robotic systems in real-time environments. *Int. J. Robot. Autom.* **21**(2), 82–90 (2006)
19. Niemann, R., Marwedel, P.: An algorithm for HW/SW partitioning using mixed integer linear programming. *Des. Autom. Embed. Syst* **2**(2), 165–193 (1997)
20. Osterling, A., Benner, T., Ernst, R., Hermann, D., Scholz, T., Ye, W.: *Hardware/Software Co-Design: Principles and Practice*. Kluwer Academic Publishers, Dordrecht (1997)
21. Quinn, H., Leeser, M., King, L.: Dynamo: a runtime partitioning system for FPGA-based HW/SW image processing systems. *Spec. Issue J. Real-Time Image Process.* **2**(4), 179–190 (2007)
22. Raja, Y., McKenna, S.J., Gong, S.: Tracking and segmenting people in varying lighting conditions. In: Proc. 3rd Int. Conf. on Automatic Face and Gesture Recognition, pp. 228–233 (1998)
23. Rohr, K.: Towards model-based recognition of human movements in image sequences. *CVGIP Image Underst.* **59**(1), 94–115 (1994)
24. Rowley, H.A., Baluja, S., Kanade, T.: Neural network-based face detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **20**(1), 23–38 (1998)
25. Saha, D., Mitra, R.S., Basu, A.: HW/SW partitioning using genetic algorithm. In: Proc. of 10th Intern. Conference on VLSI Design, pp. 155–160 (1997)
26. Scalera, S.M., Vazquez, J.R.: The design and implementation of a context switching FPGA. In: IEEE Symposium on FPGAs for Custom Computing Machines, pp. 78–85 (1998)
27. Scanina, L., Ruzicka, R.: Design of the special fast reconfigurable chip using common FPGA. In: IEEE Proc. of Design and Diagnostics of Electronic Circuits and Systems, pp. 161–168 (2000)
28. Viola, P., Jones, M.: Robust real-time object detection. In: Proceedings of the Second International Workshop on Statistical and Computational Theories of Vision (2001)
29. Walker, S.S., Brennan, R.W., Norrie, D.H.: Holonic job shop scheduling using a multiagent system. *IEEE Intell. Syst.* **20**(1), 50–57 (2005)
30. Wiangton, T., Cheung, P.Y.K., Luk, W.: Comparing three heuristic search methods for functional partitioning in hardware–software codesign. *Des. Autom. Embed. Syst.* **6**(4), 425–449 (2002)
31. Wood, M.R.: *Multiagent Systems Engineering: A Methodology for Analysis and Design of Multiagent Systems*. PhD thesis, Air Force Institute of Technology (2000)
32. Zheng, Y., Meng, Y.: Adaptive object tracking using particle swarm optimization. In: IEEE International Symposium on Computational Intelligence in Robotics and Automation, Jacksonville, Florida, USA (2007)