Chapter 1

Intelligent Embedded Software: New Perspectives and Challenges

Fateh Boutekkouk, Ridha Mahalaine, Zina Mecibah, Saliha Lakhdari, Ramissa Djouani and Djalila Belkebir

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.72417

Abstract

Intelligent embedded systems (IES) represent a novel and promising generation of embedded systems (ES). IES have the capacity of reasoning about their external environments and adapt their behavior accordingly. Such systems are situated in the intersection of two different branches that are the embedded computing and the intelligent computing. On the other hand, intelligent embedded software (IESo) is becoming a large part of the engineering cost of intelligent embedded systems. IESo can include some artificial intelligence (AI)-based systems such as expert systems, neural networks and other sophisticated artificial intelligence (AI) models to guarantee some important characteristics such as self-learning, self-optimizing and self-repairing. Despite the widespread of such systems, some design challenging issues are arising. Designing a resource-constrained software and at the same time intelligent is not a trivial task especially in a real-time context. To deal with this dilemma, embedded system researchers have profited from the progress in semiconductor technology to develop specific hardware to support well AI models and render the integration of AI with the embedded world a reality.

Keywords: embedded systems, embedded software, Codesign, intelligent embedded systems, intelligent embedded software, artificial intelligence

1. Introduction

Embedded systems (ES) [1] are changing our daily life. They are commonly found in consumer electronics, games, telecommunication, industrial, control, automotive, aeronautics and military applications.

ES development represents a hot topic in both academic research and industry. Contrary to conventional information systems, ES design needs software and hardware to be designed in a concurrently synergistic fashion, so the system functional/nonfunctional requirements are met. This new style of design is called Codesign [2]. Codesign is a collaborative and creative task requiring some specific skills in hardware, software and system engineering.

Semiconductor technology evolution (Moore law) pushes ES to be implemented as System On Chip (SOC) where all system functional elements or components are integrated in only one chip. Under time-to-market pressure, special customer requirements, rapid technologies changing, increasing applications complexity and diversity of design styles, methodologies and associated tools, ES designers must be assisted along the design process efficiently and interactively in order to minimize the cost of development and increase the productivity; many concepts have been borrowed from the software community, especially higher levels of abstraction, knowledge and experience reuse, project planning, cost/risk estimation and so on.

In contrast to traditional embedded systems which are central, simple, closed and reactive, nowadays embedded systems are becoming more complex, more autonomous, more open, more networked and more intelligent. For instance, they can execute very complex intelligent tasks to help invalid and aged persons in their daily activities with minor human intervention. These new features have pushed researchers and ES specialists to tune some well-known intelligence methods and paradigms. Consequently, a new class of ES called intelligent embedded systems (IES) has emerged. IES are discussed in some detail later in Section 6.

This book chapter puts the light on what we call intelligent embedded software.

First, we summarize all the specificities and the basic concepts which are related to traditional embedded system. Our focus is on embedded software models of computation and design methodologies. After that, we motivate the passage from embedded systems to intelligent embedded systems. Next, we define precisely what intelligent embedded software is and we discuss the possible models and approaches that can be used to model intelligent embedded software especially multiagent systems, expert systems, neural network, fuzzy logic, ontologies, bioinspired heuristics and hybrid models. We pass rapidly on organic computing. Finally, we present a possible intelligent embedded system design flow and present the main challenges and some future perspectives.

2. What are embedded systems?

An embedded system is a system that contains application-specific hardware and software suited to a particular task that is part of a larger system that is not necessarily computer (e.g., electronic, mechanical, electrical and so on). ES interact with the outside world via the sensors/actuators and are subjected to strict spatial, temporal and energy constraints. Indeed, ES are heterogeneous in nature. They typically combine software components (general-purpose processors, digital signal processors, etc.) and hardware components (ASIC, FPGA). Unlike a hardware implementation, a software implementation has the advantage of providing flexibility (i.e., the possibility of reprogramming), but at the price of satisfying performance constraints. ES are called real time if it is able to meet its timing constraints.

3. What is embedded software?

The principal role of embedded software (ESo) is not the transformation of data as in conventional software, but rather the interaction with the physical world. It executes on machines that are not, first and foremost, computers. They are cars, airplanes, telephones, audio equipment, robots, appliances, toys, security systems, pacemakers, heart monitors, weapons, television sets, printers, scanners, climate control systems, manufacturing systems and so on [3]. Traditionally, embedded software consisted of simple device drivers with or without an operating system support. ESo functions are activated by external controls, either external actions of the device itself or remote input. Embedded software varies in complexity as much the devices it is used to control. But with an increasing demand for wired and wireless communication, embedded software has started to use middleware to hide the implementation details of low-level communication.

Now, embedded software is becoming a large part of the engineering cost of embedded systems. That makes embedded software a likely place to look for engineering efficiencies and time-to-market improvements. Efficiency and time-to-market improvements come from good methodologies, good tools and talented programmers.

4. Embedded system architectures: state of the art and practice

The rapid evolution in the semiconductor technology led to the emerging of a new paradigm called System On Chip or SOC. The SOC can typically include a collection of heterogeneous processing elements such as embedded processors (RISC) for general purpose usage, microcontrollers for control-oriented processing, DSP for digital signal processing, ASIC to implement specific optimized processing, FPGA to implement reconfigurable computing, on-chip memories, analog part, RF part for wireless communication, an on-chip communication infrastructure such as buses, crossbars, buses hierarchies or a micronetwork, diagnostic elements, power management components, specific I/O interface modules and so on. The SOC can be seen as a compromise between hardware and software solutions. **Table 1** summarizes the main hardware components found in a typical SOC. **Table 2** recaptures our possible classification of SOC architectures. In traditional SOC (TSOC), the SOC architecture is centered on one master Instruction Set Architecture (ISA) processor and the other components are slaves playing the role of hardware accelerators. When SOC comprises many processors, we obtain multiprocessor SOC (MPSOC); this architecture is inspired from the multiprocessor architecture in general computers. When most application functionalities are implemented as software, the bulk of processors are ISA, and the architecture is called Software SOC (SSOC). In this case, real-time operating system (RTOS) is a first class. In the extreme case, when most of the application functionalities are implemented in hardware, the bulk of processors are ASICs and the result is what we call Hardware SOC (HSOC).

With the rapid advance in reconfigurable circuits, SOC tends to integrate more FPGA; this tendency helps to create what we call RSOC or reconfigurable SOC. This class of SOCs targets rapid prototyping.

Component	Main application	Main characteristics
Embedded RISC processor (ex. ARM)	General computing	Low performance, high flexibility, low cost
Microcontroller	Control-dominated computing	High performance, good flexibility, high cost
DSP: digital signal processor	Data-dominated computing	High performance, good flexibility, high cost
ASIC: application-specific integrated circuit	Specific computing	Very high performance, low flexibility, very high cost
ASIP: application-specific instruction set processor	Specific application domain	High performance, good flexibility, high cost
FPGA	Reconfigurable computing	Good performance, high flexibility, high cost

Table 1. Typical SOC components.

SOC class	Main characteristics
TSOC: traditional SOC	One central processor (master) with many hardware accelerators (slaves)
MPSOC: multiprocessor SOC	Multiprocessors architecture
SSOC: software-oriented SOC	SOC where software implementation is the prominent part; the architecture is mainly composed of ISA processors
HSOC: hardware-oriented SOC	SOC where hardware implementation is the prominent part; the architecture is mainly composed of ASICs
RSOC: reconfigurable SOC	FPGA-based
NOC: network on chip	A microcommunication network
PPSOC: plug and play SOC	SOC with IP reuse
PNOC: photonic NOC	Photonic technology
WNOC: wireless NOC	A combination between wired and wireless communications
QSOC: quantum SOC	SOC that contains all the components needed for a quantum information processor
CSOC: chaotic SOC	Chaotic computing-based

Table 2. SOC architectures.

The shared bus architecture represents a bottleneck in performance and scalability; for these reasons, researchers in the field had resorted to the Internet technology and tailor the ISO stack to create what we call Network On Chip (NOC) that integrates a micronetwork generally with three layers (physical, linking and network) to manage the big communication traffic between processors. With this network, scalability is also improved. One of the major problems in NOC is high power dissipation due to wired communication. The WNOC or wireless NOC presents a promised solution where some communication is done wirelessly by adding some antennas and RF modules. PNOC is a particular case of NOC where photonic

technology is used [4]. QSOC [5] and CSOC [6] represent some new tendencies and refer to a SOC implementing quantum computing and chaotic computing, respectively.

5. Embedded system's Codesign methodologies: state of the art and practice

It has been emphasized that the best way to meet system-level objectives is exploiting the trade-offs between hardware and software in a system through their concurrent design. That is what we call Codesign. In the traditional ES design approach, the software/hardware teams work independently and generally the "hardware first" approach is adopted; when the hardware engineers synthesize their design, the software engineers begin to develop their software, implement and tune it to fit the hardware architecture. We can say that this style of design was imposed (the only solution) due to the lack of a unified modeling substrate supporting both hardware and software modeling at higher level of abstraction and co-simulation.

But with the advancement in system level languages, EDA and CAD tools, simulation and emulation, both hardware and software teams are able to work in a collaborative fashion and communicate from the early stages of design and consequently to reduce the cost and optimize the quality of the final product. **Table 3** summarizes our taxonomy of the most important ES Codesign approaches. CCodesign refers to the conventional (traditional) Codesign approach. **Figure 1** depicts the main activities in the CCodesign. Starting from a unified system functional specification, the flow then proceeds toward Hw/Sw partitioning where decisions on parts that should be implemented as hardware and parts that should be implemented as software are made. Many optimization algorithms and metrics can be applied at this stage. Three

Codesign methodology	Main characteristics
CCodesign: Conventional Codesign	Traditional codesign
IP-based Codesign	CCodesign + IP reuse
Platform-based Codesign	CCodesign + platform reuse
Design pattern-based Codesign	CCodesign + design pattern reuse
Codesign for reuse	Codesign to produce reusable components
MDCodesign: model-driven Codesign	Codesign + model transformation technology
AsCodesign: aspect-oriented Codesign	Codesign based on aspect engineering
Web-based Codesign	Codesign in the context of Internet
Cloud-based Codesign	Codesign in the context of cloud computing
IDE-based Codesign	Codesign using an integrated development environment
FCodesign: Formal Codesign	Codesign-based formal specifications and verifications (for critical systems)
PCodesign: Prototypic Codesign	Rapid Codesign-based emulation/simulation

Table 3. SOC main Codesign methodologies.

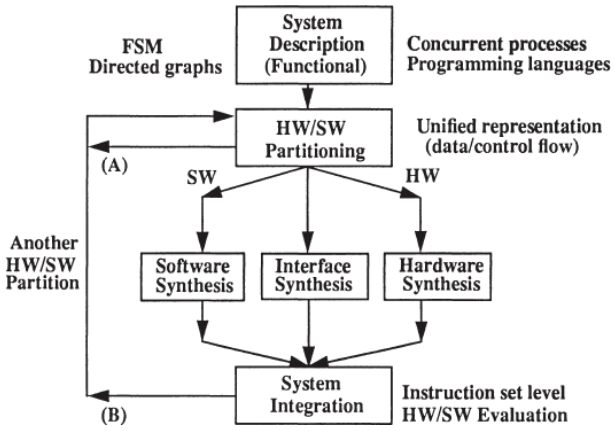

Figure 1. Conventional Codesign flow.

flows in parallel are outputted from Hw/Sw partitioning: the embedded software synthesis or compilation, the hardware synthesis, and the Hw/Sw interface synthesis. The outputs of these three flows will be then integrated for evaluation and co-simulation. As it is seen in the figure, the flow is iterative to seek for better partitions that satisfy the objectives of the design.

The system specification is the first key for the successfulness of the Codesign approach; the more the specification is expressive, complete and precise, the more the implementation will be efficient. Many requirements are identified for a good specification. Most authors prefer to use a unified unbiased model for both hardware and software. In the traditional method, such model did not exist. Embedded software is traditionally programmed in C or assembly language; such low-level languages are not portable and cannot anywhere meet system-level specification requirements. On the other hand, hardware parts were commonly specified using VHDL; but with the remarkable progress in modeling theory and programming language semantics, designers are now benefited from what we call model of computation or MOC. The latter defines formal syntax and semantic of computation and communication.

Table 4 summarizes a set of well-known untimed MOCs. Depending on the application domain, we find a collection of MOCs with different semantics. A multi-MOC denotes a MOC composing of multiple MOCs. The combination of heterogeneous MOCs is a hot research topic. Ptolemy is a good example of an environment allowing the combination of multiple MOCs hierarchically. We note the existence of a second class of MOCs called timed MOCs. The latter models and manipulates the time explicitly. As examples of timed MOCs, we find timed automata, timed Petri nets and so on.

Most existing specification/programming languages are based on one or more MOCs. We note that nowadays Codesign flows adopt SystemC [7] as the standard language for system-level specification. SystemC is an extension of the C++ language for both software and hardware programming. It supports many levels of abstraction such as transactional and RTL levels. In its earlier versions, SystemC used a discrete event simulator but with new versions it supports well other MOCs.

Model of computation	Application
DF: data flow	Data-oriented processing
SDF: synchronous DF	Data-oriented processing when the input/output size is known
KPN: Kahn process network	Deterministic data-oriented processing with infinite buffer
DE: discrete event	Discrete time processing
CT: continuous time	Continuous time processing (analog parts)
FSM: finite-state machine	Control-oriented sequential processing
DFSM: data path fsm	Control-/data-oriented processing
Statecharts	Control-oriented processing supporting concurrency and hierarchy
RS: reactive synchronous	Reactive systems with zero delay computing assumptions
Petri nets	Reactive systems (formal specification/verification)
Multi-MOC	Heterogeneous systems

Table 4. The most used models of computation.

EDA and CAD tools are also important in Codesign flow automation. Depending on the objective of the designer, we can find a plenty of tools for modeling, simulation, emulation, formal analysis, automatic code generation, optimization and verification, performances estimation, synthesis, and so on [8, 9]. The good choice of such tools may have a great impact on the quality of the final product.

The embedded software synthesis is part of the Codesign flow (**Figure 2**) [10].

Besides system-level description language (SLDL), which is able to capture both hard and software components, three major elements are needed in order to support the software aspect of the design flow:

1. Processor models that capture the processor at different levels of abstraction.
2. RTOS support for the processor.
3. A software generation tool that synthesizes user code targeted for the selected RTOS.

An RTOS provides at least the core real-time scheduling functionality, inter-task communication, timing and synchronization primitives. It is implemented and described as a real-time kernel or real-time executive. The scheduler in RTOS is designed to provide a predictable execution pattern. This is particularly of interest to embedded systems as embedded systems often have real-time requirements.

5.1. IP-based Codesign

Tends to shorten the time-to-market and minimize the cost of SOC design, the IP-based Codesign [11] emphasizes on reuse of predesigned and pre-verified components called

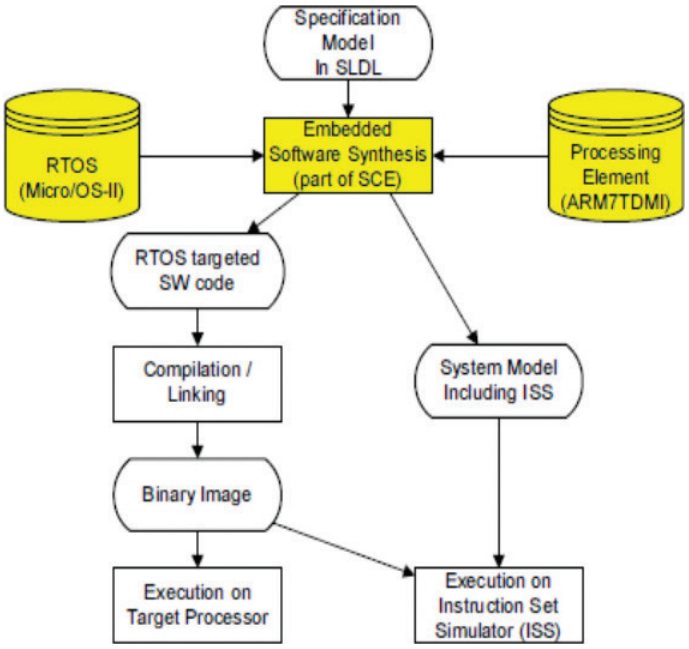

Figure 2. Embedded software synthesis flow.

intellectual properties (IPs). IPs may have many formats and specified at many levels of abstractions. In general, IPs are classified into three main classes called: soft IPs, firm IPs and hard IPs. IP reuse comes at the price of integration effort especially for incompatible IPs.

5.2. Platform-based Codesign

Instead of reusing individual IPs, this style of Codesign uses an entire platform specific for a certain application domain. The effort of design is limited to tuning the platform for the given application (bottom-up methodology) or to tuning the application for the given platform (top-down methodology). Certainly, this approach reduces considerably the effort of design, but at the price of nonoptimized designs furthermore, finding the existing platform that matches designer requirements is not trivial [12].

5.3. Design pattern-based Codesign

In software engineering, a pattern is a general repeatable solution to a commonly occurring problem. A pattern is an abstract template that needs to be refined and adapted before it can be integrated into the code. Patterns focus on descriptions that communicate the reasons for design decisions. In the field of SOC Codesign, the definition of generic patterns is more difficult. For instance, design patterns to generate wrappers for IPs integration have been already proposed and others to promote reuse beyond code reuse [13].

5.4. IDE-based Codesign

Putting a collection of tools that may be obtained from different providers and organizations in one environment to facilitate the SOC Codesign is the philosophy of the IDE-based Codesign. The main challenge in this style of Codesign is the interoperability between tools [14].

5.5. Codesign for reuse

Codesign for reuse tends to design reusable IPs. These reusable components can be soft, firm or hard, described in a standard format and well documented and catalogued for easy integration [15].

5.6. Aspect-oriented Codesign

The aspect-oriented engineering tends to increase reuse by separating in early stages between functional and nonfunctional aspects and to propose some mechanisms to integrate them lately to generate the full code. This new technology was rapidly borrowed by SOC designers [16]. This strategy will bring many advantages regarding the portability and reuse but at the price of time overhead.

5.7. Model-driven Codesign

This approach is relatively very recent and becomes very popular. It tends to apply the model-driven engineering technologies in SOC Codesign. The impetus behind this is to increase productivity by the use of a unified graphical notation (source models) to model different views of the system (functional, structural, behavioral, etc.) and the automatic transformation of such graphical notations to one or many other notations (target models). The source, the target meta-models and the transformation rules are expressed explicitly and can be used either to transform one model to another model or to refine the initial model. In this context, many UML profiles for SOC Codesign were proposed [17].

5.8. Web-based Codesign

By web-based Codesign, we refer to SOC Codesign in an Internet-based context. In this style of Codesign, designers develop their SOC online and, consequently, they can exploit available environments, tools and download what they need to accomplish their tasks. For instance, they can use sophisticated Internet research tools for IP selection, simulation and verification tools. They can also contact SOC experts and share the experience online [18].

5.9. Cloud Codesign

As a new form of Internet-based computing, cloud computing is an emerging computing style that tends to enable ubiquitous, on-demand access to a shared pool of configurable computing resources (e.g., computer networks, servers, storage, applications and services) which can be rapidly provisioned and released with minimal management effort. Cloud Codesign refers to CCodesign but in the context of cloud [19].

5.10. FCodesign: formal Codesign

Formal Codesign tends to develop SOCs implementing critical applications with hard constraints. This style of Codesign uses formal specification languages and formal verification techniques such as model checking and theorem proving to ensure the correctness of the system. The methodology itself starts from an initial formal specification and then proceeds by refinement till the code generation. Examples of such languages are B, Esterel, Lotos, Petri nets, abstract automata and so on. Generally, SOC designers are not very familiar with formal specifications requiring a deep mathematical background; for this reason, instead of dealing directly with such specifications, many tools have been developed to generate formal specification from graphical notations (UML) [20].

5.11. PCodesign: prototypic Codesign

The first objective of prototypic Codesign is to provide a rapid prototype of SOC to the customer. The prototype is generally implemented in FPGA. By exploiting existing tools of emulation/simulation, the customer requirements can be earlier validated without engaging into details. This style is very suitable when the customer requirements cannot be captured entirely in the requirements analysis phase or because the requirements change rapidly over time; in this case, SOC designers can incrementally validate the functionality using reconfigurable SOC.

6. What are intelligent embedded systems?

Intelligent embedded systems represent a novel and promising generation of embedded systems. The word “intelligent” or “smart” may imply many things: for instance, it can imply the ability to make decisions, the capability of learning from external stimuli, adapting to changes or the possibility of executing computationally intelligent algorithms.

In this context, we will define an IES as a conventional ES with the capacity of reasoning about their external environments and adapt their behavior accordingly. IES have some main characteristics such as self-learning, self-optimizing and self-repairing (**Figure 3**).

A good example where IES can be found is robotics. Robotics are basically intelligent machines whose functionality is controlled by embedded systems. Robotics contain embedded systems at their heart to perform the functions required for them, for example, pick and place systems in manufacturing industry, welding robots used in automobile assembly, and so on. Elmenreich [21] identified some potential reasons for employing an intelligent solution for embedded systems among them, such as dependability, efficiency, autonomy, easy modeling, maintenance costs and insufficient alternatives. Beyond these reasons, we can say that the first impetus behind IES is to render the human life easier, more comfortable and more secure. For instance, IES are now present in what we call smart homes, smart cities, Internet of things (IoT) and so on. IES can execute intelligent real-time tasks to manage power and water, help aged and invalid persons in their daily activities, and control smart cars and drones and many other smart devices. The presence of IES in our life becomes a necessity.

http://dx.doi.org/10.5772/intechopen.72417

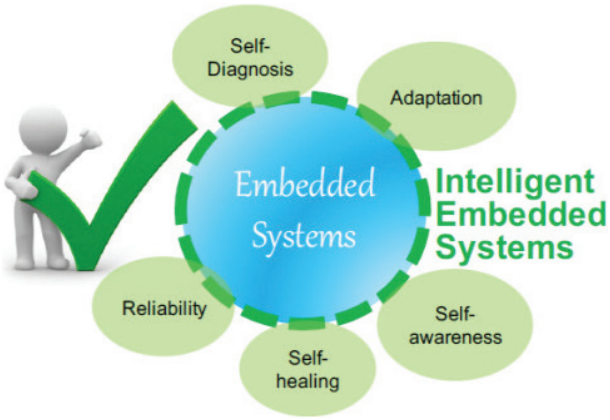

Figure 3. Intelligent embedded system features.

7. What is intelligent embedded software?

An IESo is first an ESo that has the capacity to gather and analyze data and communicate with other systems. Other criteria include the capacity to learn from experience, security, connectivity, the ability to adapt according to current data and the capacity for remote monitoring and management. As intelligent conventional software, IESo can also include sophisticated AI-based software systems, such as expert systems and other types of software. IESo exists all around us in terminals, digital televisions, traffic lights, automobiles, and airplane controls,

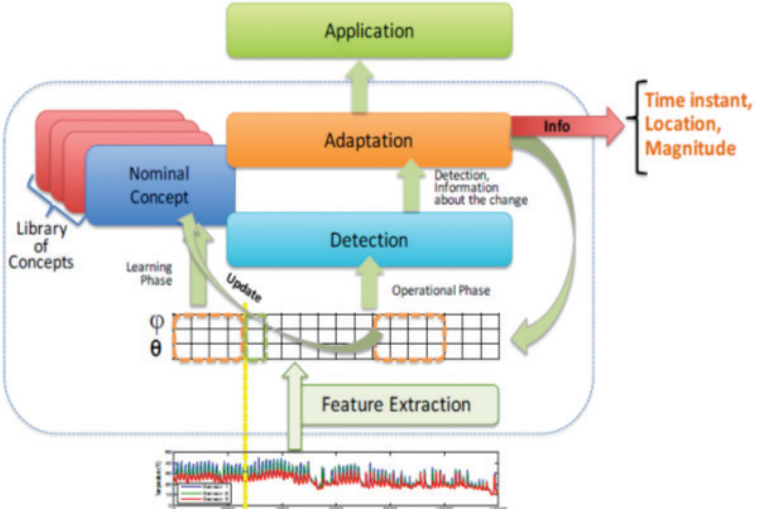

Figure 4. Adaptable embedded software flow.

among a great number of other possibilities. **Figure 4** depicts a possible flow for an adaptable IESo having the capability of learning from a dynamic changing environment. The IES have to resort to some concepts in order to interpret and comprehend the semantic of new features [22].

8. The application of AI in the field of ES Codesign

Artificial intelligence (AI) is becoming more and more attractive to model and simulate control intelligent system behavior [23]. An example is the use of knowledge-based technology for control systems that cannot be completely modeled mathematically. Recently, the use of artificial intelligence (AI) techniques in real-time control applications has emerged. In terms of embedded systems, this gives rise to the possibility of developing systems that can learn from their environment and that can change their own control programs to adapt to new situations, and these features are required to operate autonomous devices. In this section, we try to highlight some of AI methods that have been applied in the field of electronic and ES design at the same time and we will show the possibility of applying such methods in the context of IESo.

8.1. Expert systems, neural networks and fuzzy logic

The application of IA in the field of electronic design is not new. It returned to 1980s where EDA tools profited from expert system technology to assist electronic designers to make routing/placement and hardware synthesis [24].

With the ever-increasing in the semiconductor technology integration, using expert systems in IES design becomes questionable since they demand much time for reasoning, the knowledge base will be unmanaged, furthermore, rules of type if-else cannot model complex deduction process. Some works are proposed to implement expert systems in hardware, and other works are proposed to parallelize the inference process to gain time.

Expert systems have been maturated, and many environments and languages are now available to assist designers to develop their own domain-specific expert systems. Traditional expert systems are less interactive and have not the capabilities of learning; to overcome these drawbacks, researchers tend to integrate Neural Networks (NN) with expert systems, so they can learn and modify inference rules/knowledge base dynamically. Similarly, to deal with uncertain/incomplete information, fuzzy logic and some mathematical theories like rough sets have had been integrated with expert systems and NN.

In the context of IESo, fuzzy expert systems and neural networks can be applied especially in fault detection and diagnosis [25]. Cotton [26] proposed a solution for implementing neural networks on microcontrollers for many embedded applications. A new class of SOC called neural or nerve SOC implementing neural computing has been emerged [27].

8.2. Multiagent systems

Multiagent systems (MAS) have been successfully applied to model and manage complex distributed systems since they offer high capabilities for complex interactions, autonomy and reactive/cognitive behavior modeling. In the context of IES design, some authors proposed

to model intelligent agents for IP research and web-based SOC design. An IP can be soft and consequently used to execute an IESo module. Other works have applied MAS to model complex IESo, and the result is what we call embedded agents. The latter can be later synthesized as software embedded agent or hardware embedded agent [28–30].

8.3. Ontologies

Recently, the use of ontologies in software engineering has gained popularity because they facilitate the semantic interoperability and machine reasoning. Ontology is a formal representation of domain-specific knowledge. In the context of ES Codesign, some researchers, used ontologies for IP research in web semantics, for instance authors in [31], defined a VHDL ontology. The work in [32] defined ontology for IP reuse-based SOC design. The IP can be of course soft. Other works have been tried to use ontologies in the context of the Internet of things (IoT) to guarantee interoperability [33]. For example, in **Figure 4**, we can use ontologies to model the set of concepts.

8.4. Nature/bioinspired approaches

Nature/bioinspired optimization meta-heuristics has gained more attention by ES designers especially in Hw/Sw partitioning and hardware synthesis. The latter is qualified as NP-hard problems. Among bioinspired meta-heuristics, we find genetic algorithms and their variants, simulated annealing, taboo search, ant colony, PSO and so on. In contrast to exact methods, meta-heuristics is more general and aims to compromise between solution quality and search time. In the context of IESo, optimization meta-heuristics can be applied to solve the RTOS energy aware scheduling problem or jointly with neural networks.

8.5. Constraint satisfaction

In AI, constraint satisfaction is the process of finding a solution to a set of constraints that impose conditions that the variables must satisfy. Many activities are considered as constraint satisfaction problems, especially the hardware/software partitioning including allocation, assignation and scheduling problems [34].

8.6. Logic programming

The effort of designing hardware capable of supporting the declarative programming model for logic derivations can now lead to intelligent embedded designs which are considerably more efficient compared to the traditional procedural ones. For instance, Panagopoulos et al. [35] proposed an extension of the RISC architecture microprocessor for knowledge representation, based on attribute grammar evaluation, in an effort of achieving design efficiency for intelligent embedded systems.

8.7. Hybrid models

Hybrid AI models refer to the combination of the above-mentioned models. For instance, we can combine between MAS and expert systems/NN/fuzzy logic and genetic algorithms to model the cognitive part of agents.

8.8. Organic computing

Organic computer (OC) is a new emerging computing paradigm inspired from the biological organic model. It is based on the insight that we will soon be surrounded by large collections of autonomous systems, which are equipped with sensors and actuators, aware of their environment, communicate freely, and organize themselves in order to perform the actions and services that seem to be required. An organic computing system is a technical system which adapts dynamically to exogenous and endogenous change. It is characterized by the properties of self-organization, self-configuration, self-optimization, self-healing, self-protection, self-explaining and context awareness. **Figure 5** depicts the IBM's MAPE cycle for autonomic computing which is the basis for OC. Here, M is for Monitor, A for Analyze, P for Plan, E for Execute, and K for Knowledge (the autonomic element). **Figure 6** shows the OC system generic architecture which is based on the observer/controller architecture. Here, SuOC designates system under observation and control. It is composed of a set of interacting elements/agents, and it does not depend on the existence of observer/controller [36, 37].

Recently, a new class of self-adaptive SOCs emerges as a new paradigm inspired from the organic computing and especially the self-x properties. This class of SOC is called organic

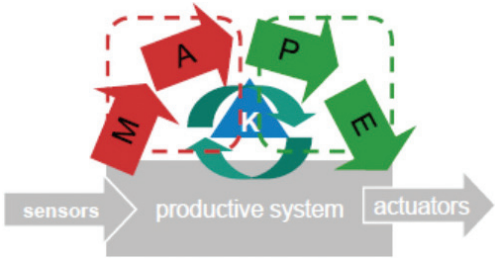

Figure 5. IBM's MAPE cycle for autonomic computing.

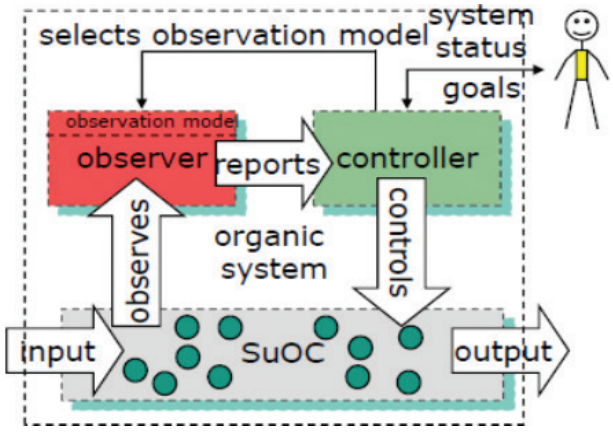

Figure 6. The generic OC system architecture.

SOC (OSOC), more suitable for smart applications having the capabilities of self-adaptation, self-control and evolvability. This new architecture is comprised of many layers and integrates more components to assure the self-x properties [38].

9. IA-based Codesign flow for intelligent embedded systems

As embedded systems become intelligent, the situation gets much more complicated regarding the application of traditional ES Codesign methodologies. In **Table 5**, we show some main differences between embedded computing and intelligent computing. The main challenge resides in how can we integrate these two styles or philosophies of computing?

In response to this aim, we propose what we call *AI-based Codesign flow for Intelligent Embedded Systems* (**Figure 7**). The idea is to enrich the conventional ES Codesign flow by another activity called *AI models partitioning* just before the HW/SW partitioning. During this activity, IES designer partitions his application functionalities or modules between a set of possible AI models such as neural network, expert system, genetic algorithm, fuzzy logic, intelligent agents, organic computing, etc., and other algorithmic modules. In other words, the objective of this step is to identify the system intelligent components and their associated AI models. At this stage, the designer can resort to some tools for modeling, simulation and formal verification. Combining AI heterogeneous models with different semantics in one framework is not an easy task and requires some validation to ensure the system correctness at higher level of abstraction. The Eclipse environment may offer an efficient solution for integration and interoperability between different AI tools and platforms. MATLAB, in turn, seems very appropriate to program and simulate some of the well-known AI paradigms such as NN, fuzzy logic and genetic algorithms. The objective of applying formal verification (i.e., model checking and/or theorem proving) at this stage is to ensure the correctness and termination properties especially in critical IES. But in order to be capable of doing formal verification, we have to define formal specification for AI models in specific formal specification languages. Depending on the used AI model, many formal languages can be employed. For example, Petri nets have been used

Embedded computing	Intelligent computing
Software and hardware are both first class	Software is first class
Resources constrained	Unlimited resources
Simple tasks	Very complex tasks
Small computing power	Significant computing power
Reactive	Cognitive
Low-level models and programming languages (Assembly, C)	High-level programming models beyond procedural paradigm
Static and completely specified environment	Dynamic and imprecise environments
Human intervention is weak	Human intervention is prominent

Table 5. Embedded computing vs. intelligent computing.

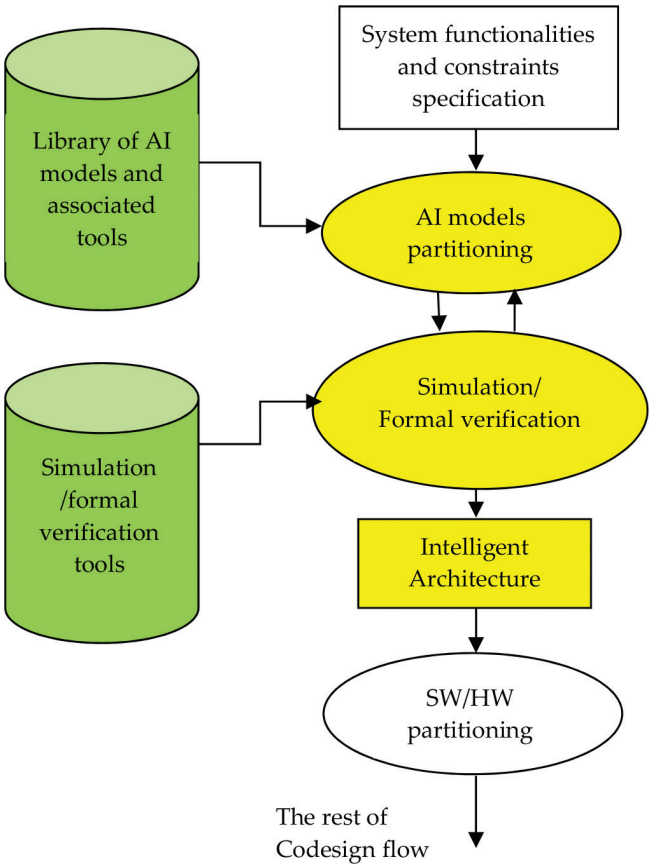

Figure 7. IA-based IES Codesign.

to formalize SMA [39]. Some authors have developed a new class of formal languages called Learning Regular Languages to formalize NN [40]. Some AI methods can be seen as formal models, especially the logic paradigm. A refinement step may also be needed to add the necessary details. The result of this step is what we call the *intelligent architecture*. After that comes the HW/SW partitioning to identify hardware/software components. For instance, we can implement a component modeled by a neural network as an ASIC, a FPGA or as a C code.

10. Challenges and perspectives

In the rapidly changing life requirements and technology, embedded software continues to dominate the values and costs of intelligent embedded systems industry. Despite the proliferation of IES over the last years, the industry of electronics and embedded systems has afraid from AI and the main question is: Can AI be a reality and apply it in IES industry efficiently without side effect?

If we know that when referring to AI, we automatically refer to human intellectual activities such as perception, learning, reasoning and memorization, self-optimization, self-adaptation and so on. The industry judgment is maybe due to the fact that the intellectual activities consume much time that can be a bottleneck for performance especially in a real-time context, where activities or tasks have deadlines or another form of timing constraints, or maybe due to the fact that AI does not reach a certain level of maturity especially at the pragmatic stage; so it can be applied efficiently in real physical systems.

For instance, multiagent systems and despite their solid theoretic basis and maturity, they are not well supported by industry. Experience from both academia and industry has proved that MAS have been used successfully to design complex, self-adaptable and even real-time systems. Currently, there are more than 80 MAS design methodologies. We think that most existing MAS methodologies in their current state are not able to deal with IES specificities; however, with some tuning and enhancement, MAS can be efficient to develop IES [41]. We note that the application of fuzzy expert systems and NN to model and simulate fault detection and diagnostics in IES is an attractive tendency. For instance, experience from both academy and industry has proved that NN have been used successfully to design self-adaptable IES.

Organic computing seems to be an attractive solution for IES but needs much effort to prove its efficiency in the industry.

In a real-time context, reasoning is known to be a bottleneck with regard to performances so in order to solve this dilemma, we can for instance parallelize reasoning or to implement it as hardware components. In all cases, we see that we must create a bridge between AI models and existing well-practiced ES Codesign methodologies and associated tools [42]. On the other hand, the progress in hardware technologies will certainly contribute in efficient implementations of IES, notably those targeting multicores and reconfigurable architectures like FPGA. Reconfigurable architectures match well dynamic and adaptable IES.

Author details

Fateh Boutekkouk[1]*, Ridha Mahalaine[2], Zina Mecibah[1], Saliha Lakhdari[1], Ramissa Djouani[1] and Djalila Belkebir[1]

*Address all correspondence to: fateh_boutekkouk@yahoo.fr

1 Research Laboratory on Computer Science's Complex Systems (ReLaCS[2]), University of Oum El Bouaghi, Algeria

2 École supérieure d'informatique (ESI), Algiers, Algeria

References

[1] Gajski DD, Vahid F, Narayan S, Gong J. Specification and Design of Embedded Systems. Englewood Cliffs, NJ: Prentice Hall; 1994

[2] Schaumont P. A Practical Introduction to Hardware/Software Codesign, 2nd ed. 2012. ISBN: 978-1-4614-3736-9

[3] Lee EA. Embedded Software. In: Zelkowitz M, editor. Advances in Computers. Vol. 56. London: Academic Press; 2002

[4] Bergman K, Carloni LP, Biberman A, Chan J, Hendry G. Photonic Network-on-Chip Design. 2014. Ebook. ISBN: 978-1-4419-9335-9

[5] Schuck C, Guo X, Fan L, Ma X, Poot M, Tang HX. Quantum Technology On A Chip. 2017. Available from: http://seas.yale.edu/news-events/news/quantum-technology-chip

[6] Ditto W, Murali K, Sinha S. Construction of a Chaotic Computer Chip. Applications of Nonlinear Dynamics; Part of the Understanding Complex Systems Book Series (UCS). 2009. pp. 3-13

[7] SystemC: www.systemc.org

[8] Mentor graphics: www.mentor.com

[9] Synopsis: www.synopsys.com/designware

[10] Schirner G, Sachdeva G, Gerstlauer A, Domer R. Modeling, Simulation and Synthesis in an Embedded Software Design Flow for an ARM Processor. Technical Report CECS-06-06 May 25. 2006

[11] Wagner F, Cesário W, Carro L. Jerraya A. Strategies for the integration of hardware and software IP components in embedded systems-on-chip. The VLSI Journal—Special Issue: IP and Design Reuse. 2004;**37**(4):223-252

[12] Sgroi M, Sangiovanni-Vincentelli A, Bernardinis F, Pinello C, Carloni L. Platform-Based Design for Embedded Systems. In: Zurawski R. editor. Embedded Systems Handbook. Print ISBN: 978-0-8493-2824-4, eBook ISBN: 978-1-4200-3816-3.2005. Chapter 22

[13] Manai Y, Haggège J, Benrejeb M. New approach for hardware/software embedded system conception based on the use of design patterns. Journal of Software Engineering and Applications. 2010;**3**(6)

[14] Carbone J. Doing embedded design with an Eclipse-based IDE. Express Logic. 2008. Available from: www.embedded.com/design/prototyping-and-development/

[15] Cavalloro P. System Level Design Model with Reuse of System IP. Kluwer Academic Publishers; 2016

[16] Deharbe D, Medeiro S. Aspect-oriented design in systemC: Implementation and applications. In Proceedings of the 19th annual symposium on Integrated circuits and systems design (SBCCI '06); 2006. p. 119-124

[17] Boutekkouk F, Benmohammed M, Bilavarn S, Auguin M. UML2.0 Profiles for embedded systems and systems on a chip (SOCs). Journal of Object Technology. 2009;**8**(1):135-157

[18] Witczyński M, Hrynkiewicz E, Pawlak A. A web services based approach for system on a chip design planning. In: Coordination of Collaborative Engineering—State of the Art and Future Challenges 5th International Workshop on Challenges in Collaborative Engineering (CCE'07). 2007

[19] Intel Labs. Single-chip Cloud Computer 2017. Available from: http://www.intel.com/go/terascale

[20] LIAMA. Formes: FORmal Methods for Embedded Systems.Visiting Committee Report. 2012

[21] Elmenreich W. Intelligent methods for embedded systems. In Proceedings of the First Workshop on Intelligent Solutions in Embedded Systems (WISES 2003); 2003

[22] Roveri M. Intelligence for embedded systems (introduction to the course) [Ph.D. and master course], Politecnico di Milano, DEIB, Italy, 2017. Available from: roveri.faculty.polimi.it/wp-content/uploads/Lecture_1.pdf

[23] Zilouchian A, Jamshidi M. Intelligent Control Systems Using Soft Computing Methodologies. CRC Press. 2001. ISBN 0-8493-1875-0

[24] Schwarz A.F. Handbook of VLSI Chip Design and Expert Systems. Academic Press Limited Harcourt Brace Jovanovich, Publishers ISBN: 0-12-632425-5. 1993

[25] Mostafa A, Elfattah M, Youssif A. An intelligent methodology for malware detection in android smartphones based static analysis. International Journal of Communications. 2016;**10**

[26] Cotton NJ. A neural network implementation on embedded systems [thesis]. Auburn, Alabama; 2010

[27] Johnson B, Lancaster K, Hogue I, Meng F, Kong Y, Enquist L, McAlpine M. 3D printed nervous system on a chip. Lab on a Chip—Miniaturisation for Chemistry and Biology. 2016;**16**(8):1393-1400

[28] Boutekkouk F, Benmohammed M. An agent-based framework for SOCs design. In Seminaire Nationale en Informatique, Biskra (SNIB'06). Algeria: Biskra; 2006

[29] Charles V. The design of a JADE-based autonomous workflow management system for collaborative SoC design. Journal of Expert Systems with Applications. 2009;**36**:2659-2669

[30] Jamont JP, Occello M. DIAMOND: Une approche pour la conception de systems multi-agents embarqués. France: Institut National Polytechnique de Grenoble-INPG; 2005

[31] Zdraveski V, Trajanov D. VHDL IP cores ontology. In: The 10th Conference for Informatics and Information Technology CIIT 2013. 2013

[32] Boutekkouk F. Towards an ontology-driven intellectual properties reuse for systems on chip design. In: Proceedings of the 2013 International Conference on Systems, Control, Signal Processing and Informatics; 2013; Greece

[33] Berrios V. Cross-Industry Semantic Interoperability, Part Three: The Role of a Top-level Ontology. 2017. Available from: http://embedded-computing.com/articles/

[34] Mitra R, Basu A. Hardware-software partitioning: A case for constraint satisfaction. IEEE Intelligent Systems. 2000;**5**(1):54-63

[35] Panagopoulos IP, Pavlatos CC, Papakonstantinou GK. An embedded system for artificial intelligence applications. International Journal of Computer, Electrical, Automation, Control and Information Engineering. 2007;**1**(4)

[36] Schmeck H. Organic computing—A new vision for distributed embedded systems. In Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05), 2005. p. 201-203

[37] Hartmut Schmeck Organic Computing—A Generic Approach to Controlled Self-organization in Adaptive Systems Institut AIFB, KIT Mars; 2009

[38] Herkersdorf A. Conquering MPSoC design and architecture complexity with bio-inspired self-organization. In: MPSoC Forum Margeaux, France, July 10. 2014

[39] Marzougui B, Hassine K, Barkaoui K. Formalism for modeling a multi agent systems: Agent petri nets. Journal of Software Engineering and Applications. 2010;**3**(12):1118-1124

[40] Madhusudan P. Learning Algorithms and Formal Verification. A Tutorial. University of Illinois at Urbana-Champaign VMCAI Nice; 2007

[41] Mecibah Z, Boutekkouk F. Comparative study between Multi Agents Systems methodologies according to intelligent embedded systems requirements. In: The 4th International Conference on Automation, Control Engineering and Computer Science (ACECS-2017); 28-30 March 2017; Tangier, Morocco

[42] Agarwal A, Shankar R, Pandya AS. Embedding intelligence into EDA tools. Series Frontiers in Artificial Intelligence and Application, Integrated Intelligent Systems for Engineering Design. 2006;**149**:389-408