Microprocessors and Microsystems 62 (2018) 20–40

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Towards Dynamically Reconfigurable SoCs (DRSoCs) in industrial automation: State of the art, challenges and opportunities

Gilberto Ochoa-Ruiz*[,a], Lina Maria Aguilar-Lobo[a], Romain Bevan[b], Florent de Lamotte[c], Jean-Philippe Diguet[d]

[a] Universidad Autonoma de Guadalajara, Mexico
[b] Lab-STICC - ComposiTIC, France
[c] Lab-STICC - UBS, France
[d] Lab-STICC - CNRS, France

ARTICLE INFO

Keywords:
Field Programmable Gate-Array
DRSoCs
Partial reconfiguration
IEC 61499 standard
Distributed control systems

ABSTRACT

In this paper we analyze the state of the IEC 61499 standard for the specification of distributed control systems (DCS). First, we discuss the limitations of previous efforts regarding the implementation of DCS, as well as the rationale for the introduction of the IEC 61499. Then, we embark in a succinct analysis of the standard and the associated models for DCS platforms, outlining the main barriers that have hindered its widespread adoption. We argue that a common architectural framework (which is currently lacking) for implementing full-fledged IEC 61499 is necessary, especially if features such as fine-grained distribution and reconfiguration are to be supported. We posit that Dynamically Reconfigurable Systems on Chip (DRSoCs) represent an excellent implementation choice for enabling such platforms,thanks to the strides made by the reconfigurable computing community in recent years, in terms of tools for implementing such systems, but also in new architectural principles and design paradigms based on Reconfigurable OSes. Moreover, we provide some compelling reasons for bringing those two domains together, as well as the challenges that need to be overcome in order to harmonize both efforts.

1. General introduction

Today's fast-changing manufacturing markets and the stringent requirements in terms of decreasing time-to-market are forcing a paradigm shift in the manufacturing processes. These tight requirements point towards an increased complexity of the *industrial environments* and the associated *control settings*, largely driven by the augmenting necessities of the current mass production markets: lean and flexible production, with the inherent high-performance requirements, such as zero defects, declining costs, increased production flexibility (for product diversification), and improved dependability, among others. Hence, in order to cope with those demands and other emerging requirements, *new manufacturing infrastructures* and production facility operation methods are very much required, as well as *new devices and embedded technologies* to support them.

One of the main challenges in the implementation of effective Distributed Control Systems (DCS) concerns the development of the *encompassing control algorithms*. It has been widely recognized that the current design practices in the development of control applications, in tandem with their deployment in centralized DCS represent the main barrier hindering the much sought improvements in the manufacturing and distributed control domains. Moreover, the *hardware control architectures* in which they are typically integrated do not lend themselves very well for the much needed *adaptability and flexibility*.

Therefore, in recent years, many research endeavors have been conducted to improve the capabilities of such manufacturing control systems, mainly based on the concept of *distributed intelligent control*. The resulting automation models are encompassed by wide networks of DCS (interconnected through *field area networks*, using fieldbuses), as depicted in Fig. 1, an approach that can potentially bring many advantages in the domain, such as modularity and decentralization, and thus increased resilience.

This novel DCS model poses very specific issues and stringent requirements in terms of the latency, reliability and availability of the of the control system and thus, it is widely accepted that traditional frameworks, based on PLC *(Programmable Logic Controller)* architectures

* Corresponding author.
E-mail addresses: gilberto.ochoa@edu.uag, gilberto.ochoa@edu.uag.mx (G. Ochoa-Ruiz), romain.bevan@univ-ubs.fr (R. Bevan), florent.lamotte@univ-ubs.fr (F. de Lamotte), jean-philippe.diguet@univ-ubs.fr (J.-P. Diguet).

https://doi.org/10.1016/j.micpro.2018.07.002
Received 13 February 2018; Received in revised form 20 May 2018; Accepted 9 July 2018
Available online 21 July 2018
0141-9331/

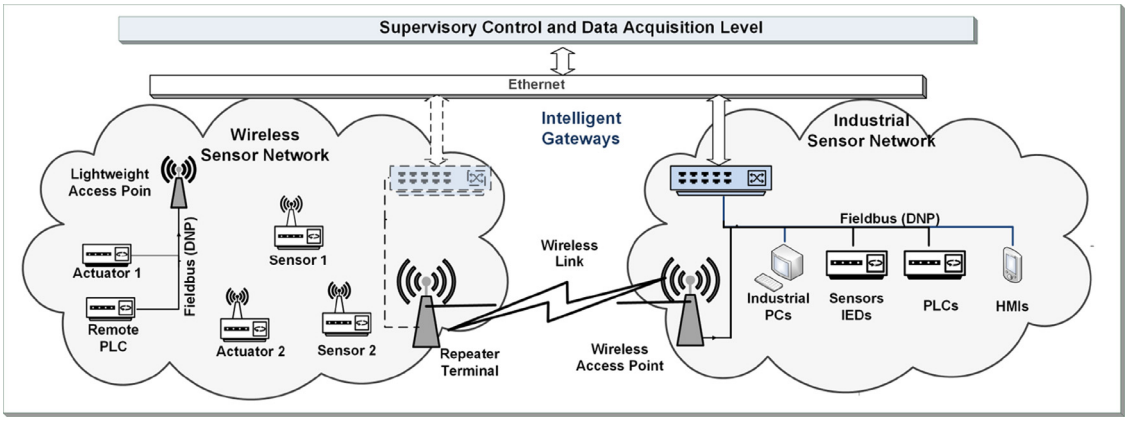

Fig. 1. Typical IPMCS hardware architecture for factory floor automation.

are not effective enough for meeting such constraints, a limitation that stems from the cycle-scan execution of the underlying platforms. These limitations have been especially evident with the advent of new automation models, such as the *IEC 61499 standard*.

Therefore, practitioners in the domain have been looking for manners to respond to these heterogeneous and often contradictory constraints [1]. In recent years, there has been a trend towards the creation and use of *Intelligent Electronic field Devices (IEDs)* or *Intelligent Mechatronic Components (IMCs)*, which contain a certain amount of computing power and communication capabilities [2].

In this context, complex functions could then be distributed over or allocated onto such IEDs, resulting in a *decreased processing load for the main PLCs* and a considerable reduction in the necessary bandwidth requirements of the application, making the devices smarter, and more easy to design and validate, as well as leading to ”processing at the edge” approaches. Such IMCs, with tightly integrated communication and configuration services, and coupled with *embedded control functions* could expedite system integration, whilst fostering extended features such as *run-time reconfiguration* [3].

However, a major hurdle for the interoperability of such IEDs is represented by the *plethora of communication protocols* used in current DCS [4], both for the wired and wireless sections of the plant (Fig. 1). Therefore, *new platforms* should integrate the infrastructure for supporting multiple industrial protocols, in a cost-effective manner, while enabling to cope with new developments, through the use of configurable and customizable on-demand communication gateways [5].

These *configurable and intelligent gateways* [6] , in tandem with customizable and reconfigurable platforms, could bring many benefits for the implementation of full-fledged IEDs, in particular those advocated by the IEC 61499 standard, as we will thoroughly argue in this article. Indeed, current methodologies and tools are regarded as too limited by many specialists, since they impose the use of *platform-specific and fixed interfaces* for implementing events and data transfers over the industrial Ethernet network and to communicate with the *I/O subsystem of the sensors and actuators they service*.

In this paper, we posit that using FPGA devices, in tandem with advanced partial reconfiguration techniques, could coalesce into an effective implementation choice for creating highly configurable, customizable and adaptable DCS and of the constituent IEDs. Such systems, capable of mixing heterogeneous functionalities which can be dynamically reconfigured and distributed, are henceforth referred to as *Dynamically Reconfigurable Systems on Chip (DRSoCs)*.

The rest of the article is organized as follows. First, in Section 2, we provide ample motivations for a shift in the design of DCS, based on the limitations of the current devices and programming languages. Afterwards, we present the IEC 61499 standard; we analyze some of the provisions of the standard, and discuss how the current hardware models are hampering its effective implementation. Then, in Sections 4 - 6, we provide some compelling reasons for using programmable devices and DRSoCs in the design and implementation of *IEC 61499-based DCS platforms*. Subsequently, in Section 7 we discuss what needs to be done, in terms of methods and tools, to make this convergence possible, as well as the current limitations. Finally, we conclude with some research directions that we consider worth pursuing.

2. Motivation: limitations of current distributed control systems

Most of the current industrial process control platforms, globally known as *Industrial Process Measurement and Control Systems (IPMCS)* are based on the well-known and widely used IEC 61131-3 standard [7], which was introduced by the *International Electrotechnical Commission (IEC)*. These systems are typically built around traditional PLC architectures, which are generally oriented towards centralized applications, in which several nodes retrieve data from the plant and adjust their behavior through actuators.

These solutions are not well suited for implementing complex DCS for many reasons, the most important being that existing approaches are designed under the *execution constraints* imposed by the *cycle-scan nature of the PLC devices*. This leads to significant limitations in the management of incoming events, as well as the throughput and real-time response of the underlying control software, especially if communication latencies are taken into consideration.

Recent developments in the automation domain point towards a shift in the underlying *Intelligent Control Production Systems*: they need to be thoroughly re-architected so that heterogeneous systems can be flexibly and seamlessly composed, from single machines to entire plants and distributed systems. This entails a significant shift from current practices and formalisms (i.e., programming languages, execution semantics), as well as the conception of *new technologies and devices*, as depicted in Fig. 2 (adapted from [8]).

The so-called *Agile Production Systems* approach entails many benefits in the design, configuration and adaptation of production systems, since it encourages the *encapsulation and reuse of control algorithms* and other development libraries from the automation domain through the use of mechatronic components, which can *significantly ease the integration and programming of complex DCS*. However, such *novel engineering frameworks* need to be accompanied by widely used and accepted *open design tools and facilities*, encompassed by visual editors, compilers and generators as well as platforms and validation tools. Moreover, in order to cope with the increased capabilities required by DCS, novel *Open Run-time Platforms* need to be conceived in order to support the much sought *Intelligent Technologies* described above, capable of *reconfiguration, distribution, diagnosis and fault recovery*. The latter represents not only technological shift, but also a methodological one, in which the various aspects of the design process can be holistically defined.

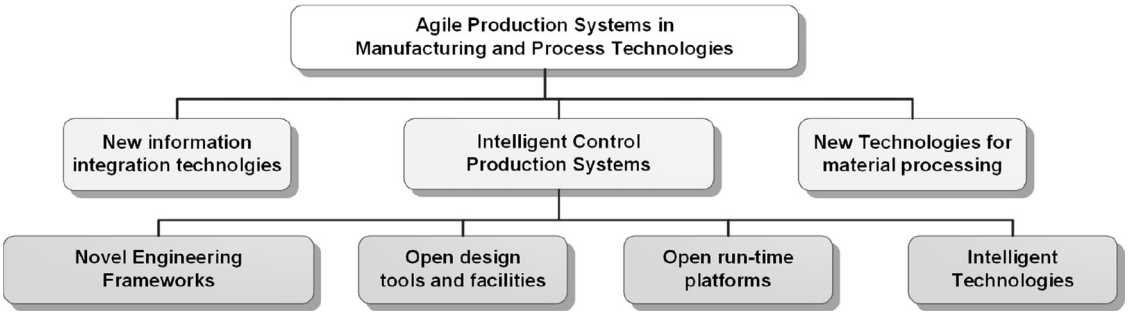

Fig. 2. Requirements for effective agile production systems, according to the OONEIDA framework.

The rigid character and insufficient agile capabilities of the current manufacturing systems can be deemed to the existing design approaches, based on *centralized architectures and hierarchical control infrastructures*. Most of these approaches are based on the well-known IEC 61131-3 standard and the associated programming languages, which although represented a successful endeavor for the convergence of PLC-based systems, is currently hindering the adoption of new paradigms.

The *main rationale behind the introduction of the IEC 61131-3 standard* was to harmonize disparate programming concepts from the industrial control domain [9], by introducing software engineering concepts into the former. A great deal of progress was made by the IEC in this regard, by defining a set of *common languages for PLC programming*, which led to a widespread adoption of the standard. However, due to extensions added by vendors, interoperability and straightforward reuse are still not completely feasible. Moreover, some of the extensions introduced a handicap in the modularity and encapsulation of the basic unit of control, the Function Block, through the use of global variables.

During the last few years, these issues have been tackled by the industry and the academia at large, specifically through endeavors such the *Intelligent Manufacturing Systems (IMS)* initiative, which aimed at defining new means for developing and accelerating the adoption of the next generation of open, modular, reconfigurable, maintainable and dependable manufacturing systems. The resulting *IEC 61499 standard* [10] for DCS was conceived as a framework for modeling *exchangeable and distributed IPMCS applications* [11].

The specification represents an evolution over the IEC 61131-3 standard to incorporate the demands of flexible and adaptive paradigms into DCS. One of the main goals of the standard is to obtain *vendor-independent application and hardware configuration descriptions*, based on the integration of control algorithms in the so-called *Function Blocks (FB)*, with improved encapsulation features.

The rationale for this encapsulation is that *it fosters reuse in application-centered modeling methodologies* (in a platform independent manner, later refined to be deployed in a given compliant hardware device). An application is represented as a network of distributed and communication FB instances that monitor signals from a *cyber-physical system through sensors, and produces events and data that are fed-back through actuators*.

Therefore, the standard was conceived as an effort to address the limitations of current IPMCS systems by promoting the advantages of DCS in terms of *modularity, ease of integration (and reuse), reconfigurability, and deployment in multiple devices (interoperability and distribution)* [12].

Another major change with respect with the previous standard is the introduction of an *event-driven approach for the execution of control algorithms embodied by Function Blocks*. This entails a larger flexibility in the scheduling of the underlying functionalities, and the use of the associated resources, as well as gains in terms of the real-time response of the individual sub-components. Some of the main concepts of the IEC 61499 standard, as well as its limitations will be discussed next.

3. The IEC 61499 standard for DCS: concepts, challenges and limitations

In the previous section, we discussed some the challenges faced by the current industrial automation methodologies, in terms of the procedural and architectural shifts that are required for addressing the much-needed agile capabilities of future manufacturing infrastructures. The IEC 61131-3 standard represented a first attempt at addressing many of the issues hindering the adoption of a common base for programming for *Programmable Logic Controllers*, but it suffers of *poor real-time performances and scalability issues* (especially in highly distributed plants), due to the *scan-based nature of the execution platforms*.

However, the most limiting aspect of the standard can be attributed to the poor encapsulation previsions for the Function Blocks (FBs), as well as the use of global variables used for data exchange among processes. These limitations impede its deployment in the creation of truly distributed and reconfigurable systems, characteristics regarded as essential in the future of manufacturing systems by the influential Iacocca Institute [13].

The Holonic Distributed Systems methodology has been introduced as a concept that encompasses the requirements for creating full-fledged distributed control systems. The approach aims at providing means for enabling the creation of highly integrated DCS, mainly through: *reusability, configurability, portability, flexibility and reconfiguration*, which have been accounted for during the specification of the IEC 61499 standard, as depicted in Fig. 3.

These requirements are briefly described as follows:

- **Portability:** The capability to port or exchange the control software in a seamless manner across different platforms.
- **(Configurability:** Any device, regardless of the vendor, can be programmed using a piece of code compliant with the standard and configured by other tools/devices.
- **Interoperability:** Embedded control systems can operate in unison to perform the overall distributed application (by using common communication protocols and other data exchange mechanisms). Several accompanying standards by the IEC have been conceived to further promote this convergence, such as the IEC 62424.
- **Reconfiguration:** The capability to dynamically modify control hardware and software functions at run-time and on-demand, without affecting the rest of the system, supporting various granularities (i.e., functions, complete device configuration).
- **Distribution:** The allocation of portions of the application (control algorithms in the form of FBs) onto different devices, with varying degrees of granularity.

The main scope of the *IEC 61499 specification* was not to produce a programming methodology per se, but to introduce first and foremost a common *model for DCS*: it can be seen then as a model-based approach with many commonalities with Model-Driven Engineering (MDE) [14]. The standard provides a common terminology, as well as concepts that

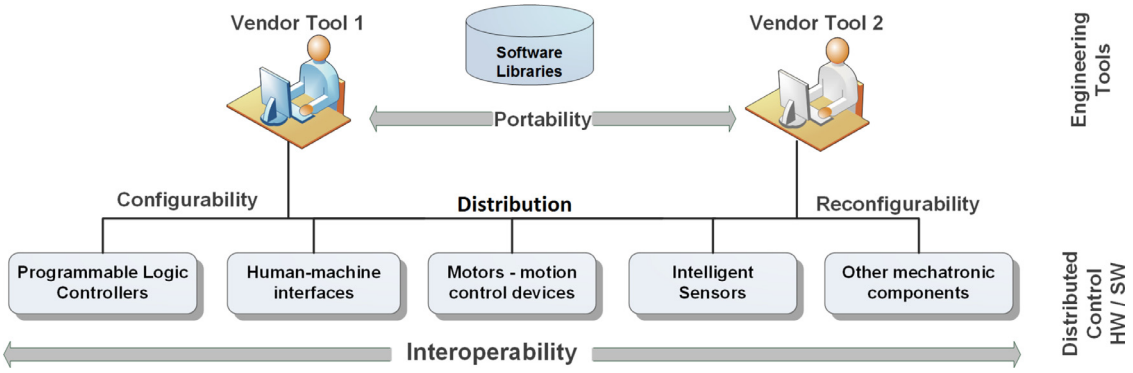

Fig. 3. Desirable features of a DCS platform based on the IEC 61499 standard.

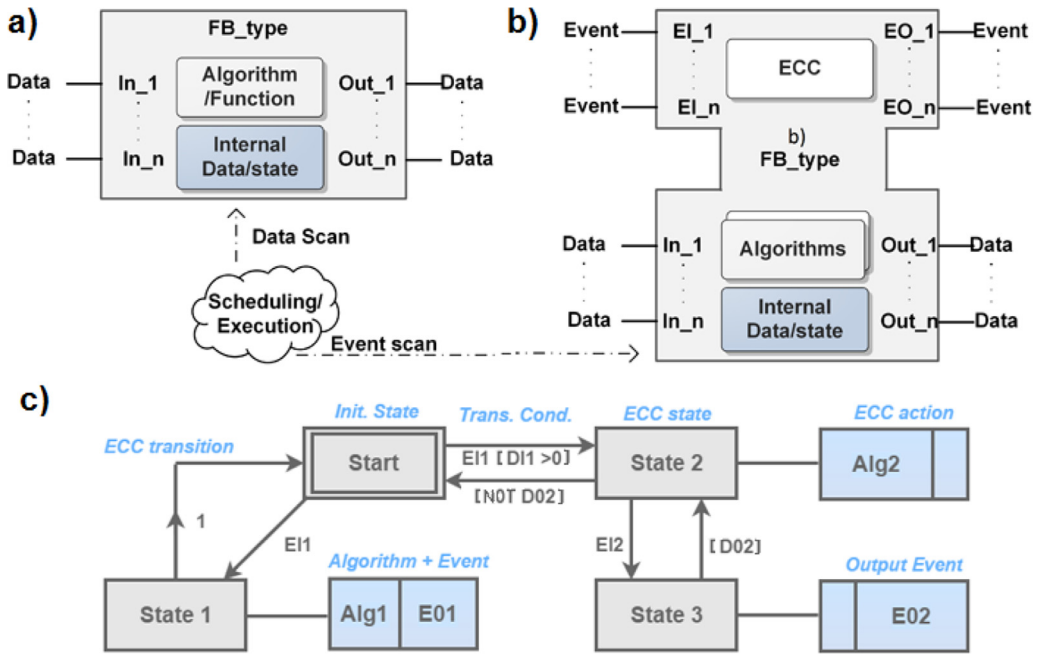

Fig. 4. Comparison between Function blocks in the (a) IEC 61131-3 and (b) IEC 61499 standards. (c) An example of the graphical representation of the Execution Control Chart associated with the IEC 61499 FB.

allow the specification of applications, where the atomic unit is the Function Block (FB) (Fig. 4). The main rationale is to enable the exchange of IPMCS platform descriptions in a formal manner [11]. In this manner, the IEC aims at encouraging the adoption of the standard and of DCS systems at large in automation applications, since the common concepts can be used to *validate, compare, reuse and exchange intellectual property of software libraries* in the form of FB types, as well as device and system configurations. As we will discuss later, the standard covers all the aspects described above, although many nuances have been *left open for interpretation (or as implementation specific options)* and have led to disparate research endeavors [15].

3.1. The IEC 61499: the Function Block concept

At the very heart of the IEC standards, it's the *function block (FB) model* that underpins the entire modeling approach (Fig. 4), upon which various programming languages and execution models have been developed [16]. An FB is described as a *functional component* with its own data structures, which can thus be accessed and modified during the application execution. A FB type definition provides a formal description of these data structures, and the algorithms to be applied to the incoming signals, which are encoded within the various instances.

The rationale for this encapsulation is that it promotes reuse in application-centered modeling methodologies (in a *platform independent manner*, later refined to be deployed in a given networked hardware system).

An application is defined as an *array of interconnected FBs* that accepts incoming signals from the *plant* through *sensors*, and produces results that are sent back to the controlled system through a feedback loop. Therefore, the standard was conceived as an effort to address the limitations of current IPMCS systems, by promoting the advantages of DCS in terms of modularity, ease of integration/reuse, reconfigurability, and deployment in multiple devices (interoperability and distribution).

In the IEC 61499 reference model, the FB encompasses a set of events, which can be concurrently propagated or processed, as depicted in Fig. 4(b), in contrast with the IEC 61131-3 FB, in which the I/Os must be scanned at the end of each cycle of the program Fig. 4(a). The main advantage of these *event-based execution approaches* is that it enables the independent execution of control processes, without the *timing penalties* associated with *scan-based automation systems*. The events enable the execution of internal, user defined processes (i.e., *control algorithms*) under the control of a federating control unit, known as the *Execution Control Chart (ECC)*.

There are four elements that define the properties and behavior of an FB: (i) the external interface declaration, (ii) internal variables, (iii)

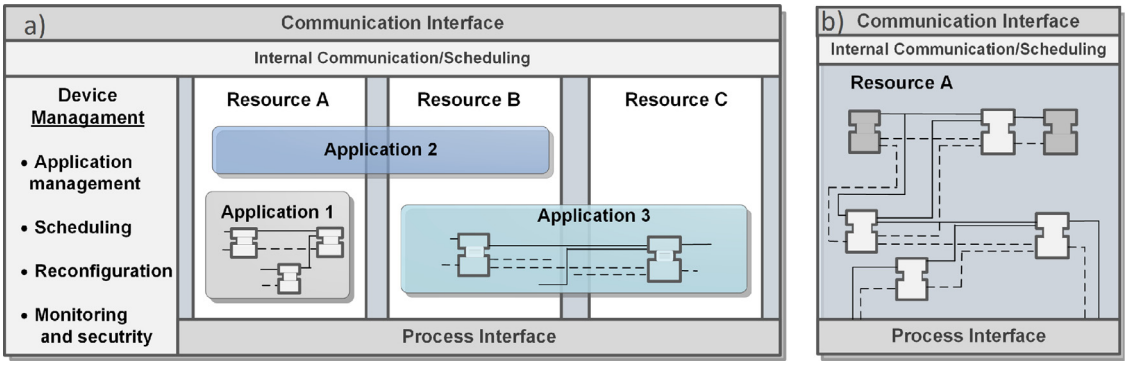

Fig. 5. IEC 61499 concepts: (a) The device model and (b) Resource model.

the internal algorithms and (iv) the execution of the latter through the transitions defined by the ECC. This is, the internal algorithms process the incoming data and produce outcomes following the state transitions specified by the ECC, which is merely a state machine in charge of federating the various stages or life-cycle of a given sub-application component, as depicted in Fig. 4(c).

The various algorithms in the FB are invoked by the resource scheduling function (to be discussed in a subsequent section) in response to a particular combination of input events arriving to the FB interface. The associated algorithm is then allowed to process data from internal variables/parameters and incoming data, which is only taken into account during the arrival of the associated event. The transitions between states in a ECC can be triggered either by an event or as a combination of an event and a guard condition.

When an FB has completed its execution, it may trigger output events to signal that output data is ready to be consumed by other FBs, either in the same device or across the DCS. Then the resource manager executive is responsible for transmitting such data to other sub-components of the application.

The IEC 61499 does not define the language that should be used for the algorithms implementation. Any *high-level language* can be used, provided that a mapping between the input and output data variables and their data types is specified; the same applies for the variables within the algorithm descriptions. These algorithmic implementations have been traditionally carried out using *Structured Text* (one of the five languages of the IEC 61131-3 standard for implementation in PLC machinery), or as in more recent implementations for general-purpose MCUs, Java or C code has been used instead. The same applies for the ECC, which can be described *textually or graphically*, but it needs to be converted to code in order to be used within the IEC 61499 context [9]. The textual syntax is rather like a built list for electronic circuits, which was developed to be used as a generic and portable algorithm description. Nonetheless, graphical representations help at making the control code easier to develop and maintain, and many efforts have been done to coalesce both efforts.

3.2. The IEC 61499 standard: application and device models

At the physical level, a DCS can be seen as a set of *devices which communicate through several networks*, comprised of various concurrent applications, which could require the cooperation of software code running in a number of *resources*. Such resources can be either within a single *device* or split among various devices located in different sites. Such application can thus be modeled as a network of interconnected FBs.

As discussed above, the standard nurtures an MDE-like approach in which the application is modeled in a *platform independent manner (PIM)* to enable exploring different design options. When mapping an application, the system designer assigns fragments of the application (Composite FBs -CFBs- or single FBs) onto different devices and resources (*Platform Specific Model, PSM*). Communication services (via the Service Interface FBs, SIFBs) and management capabilities within the device guarantee that the application is properly executed and scheduled, and caters for reconfiguration and task allocation services as well.

An *IEC 61499 compliant device*, as depicted in Fig. 5(a), should be able to support one or more resources, which are broadly defined in the standard as any entity capable of providing self-reliant execution of FBs. As shown in Fig. 5(b), the executed applications rely on the support of their *containing resources* to access the *device management services* (such as external communications interfaces and scheduling events) and to map requests to the controlled process via the associated I/O interface (*process interface*). Moreover, the system configuration might change over time, either in its entirety or by modifying functions allocated to some of the resources, which entails various scheduling policies, as well as distribution and reconfiguration granularities.

Hence, a *resource* can be seen as the *hardware underlying device* (i.e., MCU), but also as the associated *execution environment for a process* (i.e., a real-time OS). It is clear that the resource sets the boundary that exists between the provisions in the IEC 61499 standard, and what represents device specific functionality.

3.3. The IEC 61499 standard: management, execution and reconfiguration

In order to carry out the management of the applications within a device, the specification provides guidelines for the features the *device management* should have, embodied by the *device management executive*, as depicted on Fig. 5(a). The device management implements an executive application that can be seen as a middle-ware layer between the high-level control software (known as the run-time environment) and the underlying resources within the device, providing the necessary mechanisms to *exchange control commands and data* among device networks, and to federate the execution of the encompassing application FBs within the *device's resources*.

This *managerial application* should have higher privileges than normal applications, since it is indeed not only responsible of catering the communication with other *devices*, but also of the scheduling of the internal application FBs by decoding the high-level application commands, and monitoring incoming events from the *encompassing resources*. Furthermore, the device management application could also cater for other, more *supervisory, monitoring and security functions (SCADA)*, not yet defined in the standard, but which are quite important due to the increased risks in the use of interconnected *industrial control systems (ICS)*. Nonetheless, and perhaps the most important role of the device manager is that of enabling the necessary *on-line and on-demand reconfiguration* capabilities within the application.

As depicted in Fig. 6(a), the *device manager* needs to provide several services for the management of its resources, so that the *life-cycle of a given task* can be properly controlled: from its creation/allocation to its eventual removal, as depicted on Fig. 6(b). An important feature of a

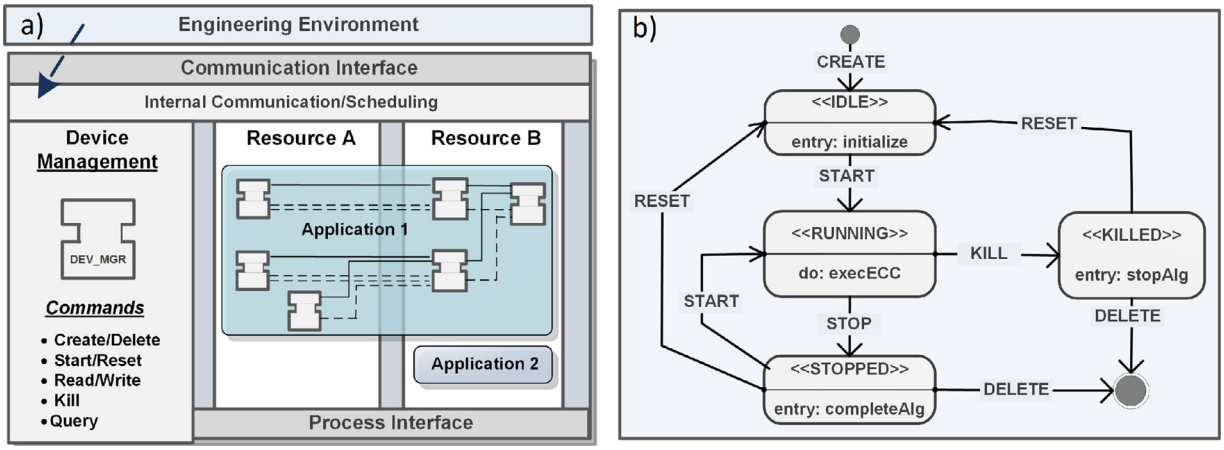

Fig. 6. (a) IEC 61499 Device management concepts and (b) Operational FSM for an execution thread .

resource is that it supports *autonomous operation*: a resource can be initialized and configured, and subsequently stopped without interfering with other resources in the same device, enabling zero-down-time, on-demand, *glitch-less reconfiguration*. Finally, in order to *schedule, load and execute* the functionalities within the a resource, a *run-time environment* must be provided [17]. The above-mentioned aspects are not yet well defined in the standard, and represent interesting research avenues to be explored.

The majority of the existing *IEC 61499 solutions* are based on *general purpose CPUs*. As with traditional PLCs, their performance suffers from the manner in which the event-based nature of the standard has been implemented (based on traditional sequential architectures): *different interpretations of the standard* have led to heterogeneous models of execution, event dispatching, and scheduling, and in general, to rather *poor real-time performances* [15]. Recent proposals in the literature [18] point at the use of FPGAs for implementing IEC 61499 platforms, with the aim of attaining *full-fledged parallel event-based systems*, with much shorter control periods and streamlined execution semantics, which has been demonstrated in other fields (i.e., motion controllers [19]).

3.4. The IEC 61499 standard: limitations, challenges and open problems

The IEC 61499 standard has been available for over a decade and has gained considerable traction within the academic community, resulting in many disparate approaches regarding the *various aspects of the reference model* (i.e., application semantics, execution polices, scheduling, reconfiguration, etc.). Several survey articles and collections have been published, pointing at the current status of the standard [20].

Most works point at the relatively slow adoption rate of the IEC 61499 standard in actual industrial applications. This should not be surprising given the fundamental changes promoted by the norm, which entails to move from *device-centric approaches* to applications running in multiple target technologies. Indeed, with the introduction of *IEDs*, the focus is no longer only on PLC devices, but on *more varied ecosystems*, encompassed by *micro-controllers* and more recently, *reconfigurable devices and architectures (FPGAs)*. Therefore, the languages that can be used to implement DCS extend beyond those in the IEC 61131-3 specification [21].

Another issue, repeatedly discussed in the literature, concerns the *open-ended nature of the IEC 61499 specification*, which was conceived more in the manner of general guidelines than a programming language, as discussed above. It has been argued that leaving *actual implementation recommendations out of the specification* standard is hindering its adoption, in detriment of the overall endeavor [22]. Furthermore, the *lack of mature reference hardware implementations*, to demonstrate both the applicability and the advantages of the DCS approach, and on the other hand, the required *learning effort* can also explain such slow adoption rate.

Several experimental academic *tools or integrated development environments (IDEs)* have been developed to facilitate the development of IEC 61499 applications, such as FBDK (FB Design Kit [23]) and the *Framework for Distributed Automation and Control (4DIAC)*[24], whereas industrial-grade tools such as ISaGRAF have been introduced recently [25]. However, these tools implement the guidelines of the standard in very different manners, first in terms of the programming languages used and secondly, in the manner the applications are simulated, scheduled and executed (i.e. the RTE) [15].

Moreover, it has been widely recognized within the automation community that a great deal of research avenues remain uncharted in regard with the provisions of the IEC 61499 standard. In particular, current tools and models do not fully support *device management applications* nor *reconfiguration modeling*. Especially, novel and more heterogeneous platforms are required to articulate the efforts in this domain, and to fully harness the capabilities sought by the standard, *enabling fully customizable and adaptable IEC 61499-based DCS systems*.

We believe that *reconfigurable devices (i.e., FPGA-based SoCs)* [26] could help to alleviate these issues, implementing the device management services, while enabling faster execution time and true distribution of FBs in the programmable logic, and adapting the communication protocol for a given scenario (and thus widening the options available for implementing several types of SIFB in hardware). The former issue could be simply solved by the specialization capabilities afforded by using reconfigurable devices, or through a more sophisticated approach, which tackles both issues. Indeed, the *use of reconfigurable devices in tandem with partial reconfiguration*, and *the associated and well researched management capabilities of DRSoCs* could have major implications in the development of IEC 61499-based platforms.

Some initial efforts have pointed at the utilization of reconfigurable devices (i.e., FPGAs) for implementing automation systems, but traditional approaches do not support reconfiguration and thus features such as distribution. Moreover, there are two major approaches for tackling such kind of implementations: they either transform IEC 61131-3 code (i.e., ladder diagrams) into full configuration images for the FPGA, or into programs to be run by multi-processor SoCs within a reconfigurable device. The former approach trades reuse for performance, whereas the latter sacrifices performance but gains in reconfiguration and distribution capabilities. Moreover, most of these systems do not fare very well regarding what type of algorithms can be implemented, because they either target software or hardened IEC 61131-3 implementations, but rarely a mix of both, which can prove useful in many instances.

Some of these efforts will be discussed in the next section, along some still very *incipient implementations of the IEC 61499 standard in FPGAs*. These efforts seek to improve the IP reuse and virtualization

aspects inherent to the standard, albeit without tackling other important features, since they do not support reconfiguration and/or distribution of FBs and resources, leading to poor implementations in terms of the consumed resources.

4. FPGAs and reconfigurable systems in automation: trends and opportunities

For quite some years, *reconfigurable platforms* such as FPGAs have been successfully used in many products, first as *ASIC replacements* and increasingly as a means of closing the gap in HW/SW development, through the combination of *high-performance processors* and *reconfigurable logic*. The use of FPGAs generally entails higher performances than microprocessor-based solutions, while ensuring a higher degree of flexibility than ASICs, due to their capability to be *updated in the field* [27].

Moreover, FPGA devices enable faster *prototyping turn-around times* compared to the latter, a fact that has increased their integration in high-end/low-volume products. In recent times, there has been an augmented interest in deploying *FPGA devices in industrial applications such as motor controllers* [28], both for the implementation of *Intelligent Electronic Devices (IEDs)* [29], and for the creation of *PLC-like solutions* [30], as well as intelligent communication gateways [31]. Thus, as we will discuss in the following sections, a harmonization of both axes is possible through IEC 61499 compliant SoC FPGA solutions, for the reasons discussed at the end of the previous section.

4.1. FPGA architecture principles

An FPGA device is an integrated circuit with a central array of logic blocks that can be connected through a configurable interconnect routing matrix. Around the periphery of the logic array is a ring of I/O blocks that can be configured to support different interface standards, as depicted in Fig. 7. This flexible architecture can be used to implement a wide range of synchronous and combinatorial digital logic functions.

This is due to the fact that their underlying fabric predominantly consists of large numbers of relatively simple programmable logic block islands embedded in a sea of programmable interconnect. In this manner, the user can map a great deal of different types of algorithms onto this reconfigurable fabric, which can be generated in a number of ways, as we will briefly discuss in the next sub-section.

Furthermore, in order to provide greater performance or flexibility in the applications that can be mapped to reconfigurable devices, FPGA vendors have gradually incorporated more functionalities into their programmable chips [32]. These functionalities are embedded among the circuit logic resources, producing a heterogeneous structure, where the capabilities of the logic cells are not the same throughout the system. This approach has become a necessity due to the widespread adoption of FPGAs for implementing complex platforms, such as Systems on Chip, where multiple processors, communication standards and memory controllers are required.

Furthermore, many applications (such as control and signal processing) often require of complex functions such as adders and multipliers, along complex and highly parameterizable DSP Blocks. In both scenarios, such modules are difficult to implement in an optimal manner using generic FPGA logic resources; therefore, FPGA vendors have introduced such complex blocks in the form of hardwired functions.

These blocks are designed to be as efficient as possible in terms of power consumption, silicon real estate, and performance. Each FPGA family features different combinations of such blocks, together with varying quantities of programmable logic blocks, which enables the designer to choose the correct device for a given application.

Another type of heterogeneous logic structures are memory blocks, known as Block-RAMs, which are scattered throughout the reconfigurable hardware. These blocks enable the storage of frequently used data and variables, and permit a quick access to these values due to the proximity of the memory inside the FPGA to the logic. These blocks can also be used for a variety of purposes, such as implementing standard single- or dual-port RAMs, FIFO functions, state machines, and so forth.

Another major trend has been a move towards the introduction of *complete microprocessor solutions into the FPGA fabric* [33]. For instance, soft-core processors (i.e. components that made use of the FPGA logic resources for their implementation) have been deployed in a variety of applications where some form of management or intelligence is required. Additionally, the use of embedded processors brings many advantages, particularly the reduction in chip count, signal integrity issues, and a decrease on the board complexity.

Similarly, as performance and throughput become increasingly important, modern FPGA devices offer support for various high-speed communication standards. This is due to the high-performance of FPGA in accelerating complex computations: data needs to be transmitted rapidly and efficiently to other nodes of the system, while preventing the device to become the bottleneck of overall the platform. Examples

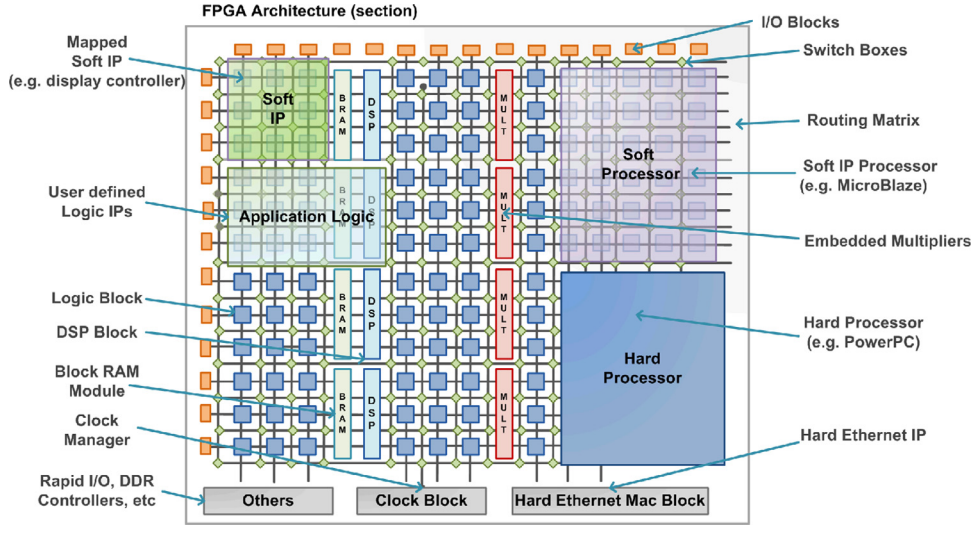

Fig. 7. Example of an FPGA architecture.

of these standards are Infiniband, PCI Express, RapidIO, and 10-gigabit Ethernet, among others. In a similar manner, vendors offer optimized hardwired cores for a variety of solutions; examples of these cores are memory controllers (for DDR, DDR2 and DDR3 type memories), MAC controllers (for implementing the physical layer of Ethernet-based communication standards), and LVDS Transmitters [34].

At the other end of the spectrum, *soft IP refers to a source-level library of high-level functions* that can be included in the design. These functions are typically implemented using or compiled a hardware description language, or HDL, such as Verilog or VHDL at the Register Transfer Level (RTL) of abstraction. Any soft IP functions the designers decide to use are incorporated into the main body of the design, which is also specified in RTL and *subsequently synthesized down into a group of programmable logic blocks* (possibly combined with some hard IP blocks like multipliers, DSPs and BRAMs) and then mapped onto the FPGA. In the next section, we will briefly discuss some of the approaches for creating and integrating such soft IP blocks into a design, since it will be helpful for the discussion that follows.

4.2. Approaches for mapping algorithms to reconfigurable logic

The evolution of computing architectures, spurred by the relentless growth in silicon integration technology, has seen the development of great number of algorithmic implementation approaches. These platforms include single chip *Multiprocessor or heterogeneous system-on-chip (MPSoC)* solutions, Networks on Chip and indeed, FPGA-based implementations of such types of architectures [35].

Modern applications implemented in FPGA-based MPSoC platforms are created using by a mixture of heterogeneous processing architectures, including Microcontroller Units (MCU), increased computationally capable processors such as VLIW DSP microprocessors, or dedicated hardware, in the form of *soft IP for efficient hardware accelerated implementations* (often as a substitute or complement to existing software-based platforms) [36]. Nonetheless, the use FPGA-based embedded platforms entails entirely different and more complex implementation issues to the designer, due to the lack of a pre-defined processing architectures and the level of expertise required for successfully implementing complex systems in reconfigurable devices.

This wide range of target processing platforms and corresponding implementation techniques makes *algorithmic implementations*, at the current levels of design abstraction, *an arduous task*. Consequently, the use of novel rapid implementation frameworks (for facilitating tasks such as code generation and function wrapping, system integration, HW/SW co-design, and system validation) is essential, and has been an active area of research for some time within the SoC and EDA communities.

As the complexity and density attainable by FPGA vendors has increased with Moore's Law, FPGAs passed from simple glue-logic devices to more complex systems for DSP applications and in packet processing in networking applications, becoming in many instances the federators and control management units of the devices in which they are integrated, and showing significant speed-ups in many applications.

This substantial jump in performance required similarly substantial advances in FPGA design tools [34]. The key feature of HDL-based design flows is their use of logic synthesis technology, as depicted in Fig. 8, which began to appear on the market around the mid-1980s. *FPGA tools traditionally take as input an RTL representation of a design*, along with a set of timing constraints.

The logic synthesis process automatically transforms the RTL code into a mixture of registers and Boolean equations, performs various optimizations (i.e., area and timing), and then generates a gate-level netlist that should meet the specified timing constraints. Subsequently, *technology-specific* placement and place-and-route algorithms are deployed for generating a physical implementation of the original specification, which can then translated into an *actual bitstream for configuring the device*. An alternative process might consist in packaging the RTL implementation using a bus wrapper, with the associated metadata for IP reuse in SoC-based designs.

Despite the success of HDL-based FPGA implementations, the *associated design process has some major drawbacks* [37]: (i) Capturing and verifying RTL design is time-consuming, (ii) evaluating alternative implementations is difficult, since it requires a long and time-consuming re-design process, (iii) accommodating specification changes and evolving a design can be rather difficult and finally, (iv) RTL specifications are often implementation specific (retargeting a complex design represented in RTL from one implementation technology to another can be a complex process).

Furthermore, the traditional FPGA design flow, based on traditional HDL specifications *is less than ideal for hardware-software co-design strategies* in SoC implementations. Irrespective of whether these designs are to be realized using ASICs or FPGAs, today's SoCs are exhibiting an ever-increasing amount of software content [38]. When coupled with increased design reuse on the hardware side, in many cases it is necessary to verify the software and hardware concurrently so as to completely validate such things as the system diagnostics, RTOS, device drivers, and embedded application software. Generally speaking, verifying the underlying hardware represented in VHDL or Verilog, in conjunction with the software described in C or assembly language is becoming a very complex process to manage efficiently.

FPGA-specific system synthesis problems are numerous. Three concerns are most prominent for the SoC community at present: *synthesis of dedicated hardware intellectual property (IP) cores* and their integration in MPSoCs; *software synthesis for multiprocessor architectures* which caters for HW/SW co-design practices, and finally *large-scale system level design and fast prototyping techniques* for deriving and automatically realizing a given application onto an heterogeneous FPGA architecture.

One of the approaches in vogue in recent years has been the use of *high-level languages (HLLs)*, as well the introduction of the associated compilation techniques to facilitate the design process of SoC platforms in general and of FPGA-based architectures in particular, by facilitating the mapping of domain-specific applications into reconfigurable devices, closing the gap between software and hardware development and shortening the design process [39].

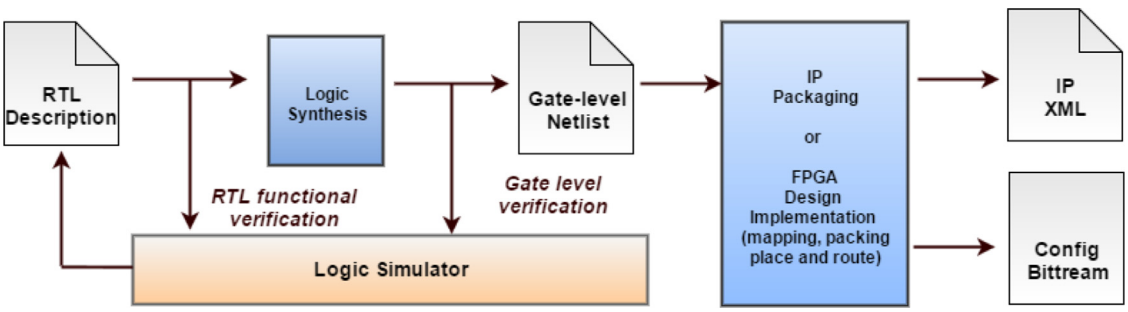

Fig. 8. Basic HDL-based design flow for FPGA implementation.

Controller / Test Algorithm	2 PID Controllers		LD with 8 timers		LD + PID + 8 timers	
	t [us]	Speed-up Factor	t [us]	Speed-up Factor	t [us]	Speed-up Factor
Sismatic S7 319	498	**1**	944	**1**	1442	**1**
PLC on FPGA [Chmiel, 2015]	540	**0.922**	1120	**0.843**	1660	**0.869**
FPGA LUT-based PLC	100	**3.56**	100	**9.44**	100	**14.42**
FPGA DSP48-based PLC	66.7	**7.46**	126.7	**7.45**	200	**7.21**

Fig. 9. Various implementations of IEC 61131-3 algorithms into FPGA.

During the 1990s, the first generation of commercial *High-Level Synthesis (HLS)* tools was made available commercially. Around the same time, research interest on hardware-software co-design, including estimation, exploration, partitioning, interfacing, communication, synthesis, and co-simulation has gained momentum, as thoroughly discussed in [40]. Furthermore, the concept of *IP reuse and platform-based design* started to emerge in the 2000s, with a shift to an electronic system-level (ESL) domain that promises to improve the exploration, synthesis, and verification of complex SoCs.

One of the major trends in ESL is the use of *HLS techniques for implementing a variety of complex algorithms into hardware* [41]. HLS tools transform an untimed high-level specification into a fully timed implementation. They automatically or semi-automatically generate a custom architecture to efficiently implement the specification. In addition to the memory banks and the communication interfaces, the generated architecture is described at the RTL level and contains a datapath and a controller, as required by the given specification and the design constraints. Such specification can be further packaged and wrapped using bus interfaces for promoting IP reuse, facilitating system integration and fostering HW/SW co-design.

The principles behind HLS techniques, as well as successes, advantages and disadvantages of such approaches are out of the scope of this article, but the reader is directed to excellent sources in this domain [42]. Nonetheless, in the next section we will explore some incipient attempts at exploiting HLS techniques and in some instances, the MPSoC paradigm for implementing automation systems in FPGA, based both in the IEC 61131-3 and IEC 61499 standards, as well as their limitations.

4.3. Initial efforts for implementing algorithms based on the IEC norms into FPGA

The study of the possibilities of FPGAs for implementing PLC-based systems has been a dynamic area of research [43]. The most important benefits of using reconfigurable devices for implementing PLC platforms are related to performance, both in terms of the execution speeds that can be attained by *massively parallel architectures*, in tandem with significantly larger I/O processing capabilities, which have been regarded as essential for overcoming the limitations of *cycle scan-based solutions*.

Several approaches have been proposed in the literature, which we have broadly divided in the next two sections: the former analyses efforts for transforming IEC 61131-3 into synthesizable code for PLC in FPGA implementations, whilst the latter discusses incipient implementations of IEC 61499 systems in programmable devices.

4.3.1. Implementations of the IEC 61131-3 norm in FPGAs

In recent years, several tools and design methodologies have been developed, which aimed at providing means to transform IEC 61131-3 algorithms (based mostly on Sequential Function Charts -SFC- or Ladder Diagrams -LD-) into executable code, which can be directly deployed into custom architectures running in FPGA devices.

From the existing literature, we can distinguish two main architectural approaches to implement such kind of devices: the first one tackles the problem by transforming IEC 61131-3 code into HDL code, whereas the second takes advantage of the possibility to implement multiple processors into a single FPGA device and the availability of hundreds of I/O signals to expedite the application performance.

The former approach can be indeed lumped to other ongoing efforts in applying HLS techniques to the design and implementations of complex control algorithms in FPGAs, with the aim of shielding the designer of domain and technology specific aspects. Some incipient solutions can be found in [44], where the authors describe a methodology for *converting PLC programs* into Verilog code, whereas the work of Ichikawa et al. [45] caters for the *generation of FPGA systems from LD diagrams*.

A more comprehensive and formal approach to this problem has been presented in [43], where the authors make use of a language-independent PLC program representation as a passage between LD and SFC programs and hardware functions in the FPGA, based on an *Enhanced Data Flow Graph (EDFG)* approach for HLS. Afterwards, the control algorithms undergo a series of optimization techniques for implementing a variety of functions, either in general purpose logic (CLB and LUTs) or using specialized DSP48 blocks of the FPGA. Several experiments have been carried to demonstrate the suitability of this latter approach; for instance, the authors present the implementation of various LD and SFC-based programs encompassing logic, timers and PID controllers and combinations thereof as depicted on Fig. 9.

We focus on the results of Milik [43] mainly due to space considerations, but the results shown here, and those presented in other works clearly demonstrate *significant speed up factors compared to algorithms implemented in traditional PLCs* (Sismatic S7) and on soft-processor PLCs in FPGA [30] to assess the effect of the various implementation options, as it can be seen in the last two lines of the table, trading resource utilization for performance.

Despite their appeal, the systems and design methodologies described above (*essentially small FPGA-based PLC on Chip* [46]), represent centralized solutions, using mostly digital I/Os and furthermore, and can be seen as all or nothing solutions, in the sense that the generated code is used for programming the totality of the FPGA. Moreover, such approaches pose other issues, which stem from the *poor encapsulation of the deployed algorithms* and the manner they are implemented into hardware. First of all, these approaches do not foster IP reuse or take advantage of the inherent programmability of the FPGA, and therefore, adapting them to new applications even if small changes are required is rather cumbersome. These limitations also hamper features necessary for implementing agile DCS, such as fine-grained distribution and reconfiguration, issues that arise from the poor encapsulation strategies attained by efforts such as those discussed above.

Thus, a second approach for implementing PLC-like functionalities around the IEC 61131-3 standard in FPGA devices has recently caught momentum, based on the use of a *Multi-processor Systems-on-Chip (MPSoC) paradigm*, which aims at improving the programmability of FPGA-based solutions, as well as the portability of the generated code, aided in many instances by co-processing units for implementing bit-wise operations [46]. Examples of this kind of approach for implementing a single micro-PLC can be found in [30], which runs LD code compiled to the processor architecture. More recent works [47] present *a design and compilation environment for multi-processor architectures*, which enable the authors to translate various forms of IEC

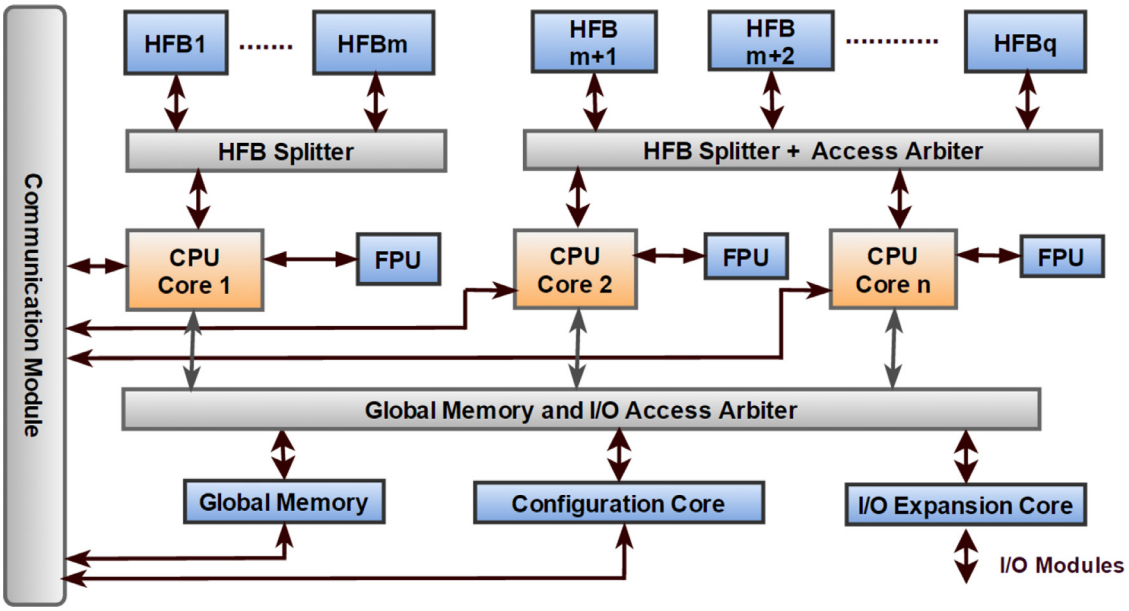

Fig. 10. MPSoC-based PLC architecture targeted to FPGA devices [47].

61131-3 control algorithms, either in the form of executable code to run in the custom processor, or as custom hardware accelerators, as depicted in Fig. 10.

This architecture is encompassed by a scalable FPGA-based MPSoC architecture, consisting of a number of individual soft-processors (designed and optimized for running IEC 61131-3 code though a compilation chain that generates target-independent virtual machine code), which can include specially designed *floating-point units (FPUs)*, for intensive and resource costly operations. The platform integrates a communication interface for configuring the SoC and the various peripherals, which include a programmable I/O expansion core. SD and SFC code can be mapped to the various processors, which can work either independently or in coordination to implement a given application. The processor solution alone has been demonstrated to improve the performance of various control algorithms, such as LD, PID and more complex designs (a fuzzy rule-based system dubbed as P1-TS by the authors) as shown in Fig. 12, when compared to a plethora of PLC-based solutions, even using a single processor. One of the advantages of the approach is that *it lets the designer to transform complex algorithms into hardware function blocks*, which is useful for accelerating computing intensive functions of the application.

The latter implementation choice, dubbed by the authors as *Hardware Function Blocks (HFBs)*, are generated much in the same manner as in the methodologies discussed above (converting IEC 61131-3 software to HDL), but the obtained code is wrapped to foster re-usability and thus facilitate the integration of complex systems. The results shown here (in summarized form), and depicted in the last line of the table, indicate that the execution times of some computation intensive algorithms can be significantly shortened, and the use of MPSoCs in this particular context could enable to carry out various HW/SW co-design strategies and design exploration techniques if coupled with appropriate profiling tools. It is important to note the these kind of approaches, as those advocated by the IEC 61499 standard, have been seen by many researchers as a means to overcome the performance bottleneck in many domains, since it could help unleash the potential of full-fledged heterogeneous multi-core architectures for a plethora of applications [48].

Nonetheless, and despite the fact that this *MPSoC approach improves significantly the programmability and encapsulation issues* of other pure hardware implementation of IEC 61131-3 design frameworks (hence promoting the distribution and reconfiguration of the software tasks of a given application), the HFB portions of the architecture still suffer from the handicaps of the design flows described above: if even a single hardware task (HFB) is to be modified or relocated, *the entire design would need to pass the entire FPGA design and implementation phases*, making more difficult the verification process as well. Another issue is that these *hardware functions need to be present in the reconfigurable fabric during the entire application lifetime*, increasing the resource utilization and power consumption in the FPGA. As we will see, the same problems plague some initial implementations of the IEC 61499 in reconfigurable devices, issues that could be alleviated through partial reconfiguration techniques and Dynamically Reconfigurable systems on Chip (DRSoCs).

4.3.2. Initial efforts for implementing FPGA-based IEC 61499 solutions

As with the systems discussed in the previous section, in recent years there has been an increased interest in implementing automation and control systems in FPGA, based in the IEC 61499 norm. As we have thoroughly discussed in Section 3, the specification provides means for modeling DCS, as well as general guidelines on how such systems should operate to integrate the provisions of the standard. However, the actual implementation features that the hardware platforms should include have been left out of the norm [49], as well as other more managerial aspects related to device and resource managers. The only constraint is that the Functions Blocks should follow the execution semantics briefly described in Section 3.2, regardless of the underlying device management and HW implementation.

Despite the inherent advantages of reconfigurable devices, very few endeavors have targeted FPGA-based IEC 61499 systems, the work of O'Sullivan and Heffernan [18] being the most prominent so far. The authors describe various *custom logic blocks* necessary for implementing *IEC 61499 applications onto FPGA circuits*. In particular, the authors focus on the implementation of the *resources and FB models using HDL descriptions*, which are shown in Fig. 11 (a) and (b), respectively.

The authors provide a thorough description of the implementation of these two models of the IEC 61499 standard, which encapsulate the basic features of the models as bus-based hardware accelerators written in HDL and optimized for FPGA implementation. The resource implementation, depicted in the left side of Fig. 11 implements the communication infrastructure with the device through specialized receiver (RX) and transmitter (TX) FBs, which in tandem with the resource manager, federate the execution of the underlying FBs. The resource manager is also in charge of polling the process interface to detect any

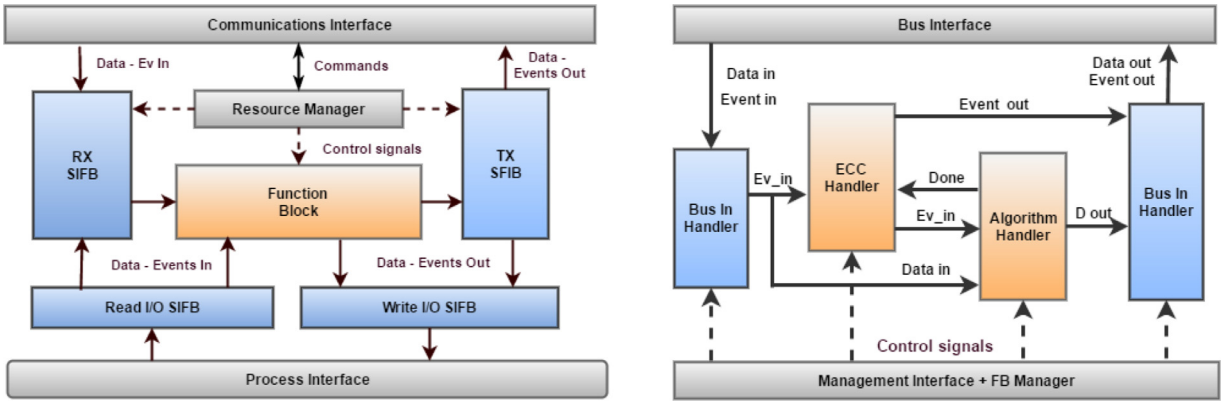

Fig. 11. Incipient FPGA implementations of the IEC 61499 for (a) a Resource and a (b) Function Block [18].

incoming data and/or events, which control the behavior of the algorithms contained in the FB; it also caters for delivering the results of the algorithms to the controlled process through an I/O interface.

The resources encapsulate the function block models, which are essentially bus-wrapped hardware accelerators that implement the basic services required for communicating with the high-level manager. As depicted in Fig. 11(b), the function blocks are wrapped by bus-handlers for writing and reading operations by the encompassing resource managers, delivering events to be used by the ECC to trigger the various algorithms encapsulated by the FB.

The ECC Handler and Algorithm Handler modules in the figure are generated though a high-level synthesis process in which IEC 61499 algorithmic descriptions are converted to VHDL, which needs to be integrated in the FB wrapper. The authors present some interesting results and case studies, but no performance analysis is provided. Furthermore, the results presented by the authors seem mostly preliminary, since no provisions are described for the *overall management of the device and the encompassing resources*. Furthermore, the *proposed architectures are inherently static*, in the sense that the totality of the original specification is mapped onto the FPGA, without taking into account the much sought advanced features of the standard (in particular distribution and reconfiguration).

As with some of the IEC 61131-3 architectures discussed before, the modification an FB within a resource entails *the reconfiguration of the entire FPGA device*, which betrays the provisions of the IEC 61499 standard. Secondly, all algorithms must be present in the FB to be triggered by the ECC, incurring in severe penalties in terms of the *occupied resources in the reconfigurable fabric* [50], much in the same manner as HFBs in the MPSoC architecture described before [47].

Furthermore, the proposed interfaces and constituent sub-components need to undergo a standardization process or at least be agreed upon by the community at large, in order to promote IP reuse and exchange within the automation community, if the efforts presented in these works are to have any lasting impact. These *standardization efforts have of course a long history in SoC design*, which have slowly coalesced into a handful of major bus architectures and interfaces, used today in the development of many complex SoCs.

As we will see in the next section, many of these standardization efforts have undergone several developments within reconfigurable computing community for some time [51], with the introduction of various *SoC-based architectures and standard hardware accelerators*. In particular, many efforts have been carried out to harmonize various research efforts to render such platforms more dynamic through the introduction of partial reconfiguration techniques, which enable to modify components of the architecture at run-time and on-demand. *Many of these efforts share many commonalities with the previsions of the IEC 61499 standard*, and many works have tackled the modeling, implementation and management of reconfigurable and distributable hardware accelerators, which is slowly coalescing into standards for DRSoC hardware blocks [52].

5. Implementing IEC 61499-based reconfigurable systems in FPGA-based SoCs: opportunities and challenges

We believe that the research in reconfigurable computing, and in particular many PR features could have a positive impact in the *development of full-fledged IEC 61499 platforms*, first as a common ground for the research community in the latter domain, and subsequently for more industrial grade applications *as the current limitations in the language provisions are overcome*. We will explore some of the benefits of bringing these two domains together in this section, the current limitations, and some of the challenges that need to be addressed methodologically and in terms of the current tools to make this possible. The next section will then provide ample evidence of recent strides in this front by the reconfigurable computing community in this regard, and how these developments could help in attaining these goals.

5.1. Dynamic partial reconfiguration of FPGA devices: concepts, tools and limitations

Partial reconfiguration (PR) was introduced originally as means *to increase the flexibility of FPGA-based applications*, and was initially geared towards enabling the designers to modify small sections of the reconfigurable fabric, in a *software-defined manner*. However, these

Controller / Test Algorithm	Ladder Diagram		PID		P1 - TS	
	t [us]	Speed-up Factor	t [us]	Speed-up Factor	t [us]	Speed-up Factor
GE Fanuc Versa - Max	446.8	**38.8**	2294	**417.1**	36700	**312.9**
Siemens S7-1200	339.8	**29.5**	122.9	**22.3**	2383	**20.3**
Beckhoff CP6607	1	**0.8**	9.4	**1.7**	139.2	**1.2**
MPSoC - PLC (1 CPU)	11.5	**1**	5.5	**1**	117.3	**1**
MPSoC - PLC (1 CPU) + HFB	0.0042	**2238**	0.44	**12.5**	10.3	**11.4**

Fig. 12. Comparison of various implementations of IEC 61131-3 algorithms into FPGA devices, as reported by Hajduk et al. [47].

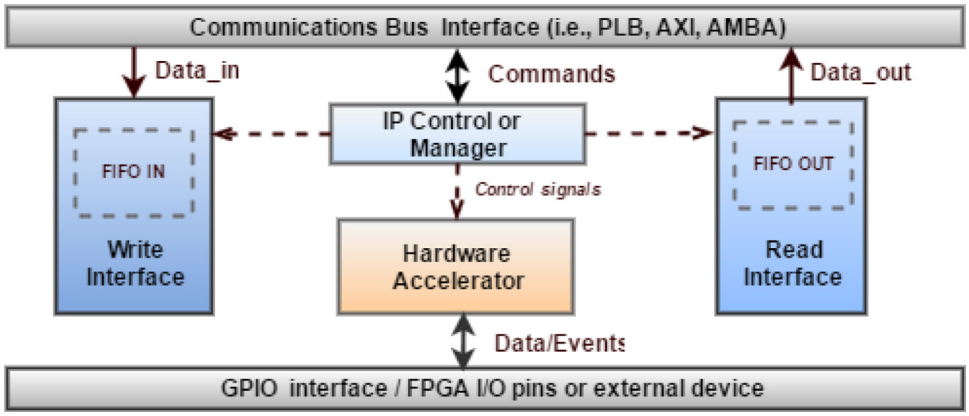

Fig. 13. A prototypical implementation of a hardware accelerator in an FPGA-based SoC.

initial implementations were rather limited since they only enabled to swap soft IPs in and out the reconfigurable fabric with relatively large granularities, and furthermore, several limitations hindered its application to more complex scenarios. Despite these disadvantages, a great deal of research has been carried out to promote the development of FPGA-based SOCs which exploit the inherent advantages of software-defined PR [53]. This term implies that a management unit in the SoC can adapt sections of the SoC (i.e. typically hardware accelerators in the form of peripherals or specialized functions) depending on internal or external factors.

These efforts are very much aligned to those in reconfigurable computing for wrapping functions in PLB or AXI interfaces, for static hardware accelerators [50]. A *prototypical hardware accelerator for a basic FPGA-based SoC* is depicted in Fig. 13; as with the Hardware Accelerators described in Section 4.3 (for implementing IEC 61131-3 algorithms hardware function blocks and IEC 61499 functions blocks, respectively), most *FPGA-based accelerators are encompassed by very similar building blocks* [54].

Nowadays, most FPGA vendors provide means for interconnecting user defined logic (i.e., the hardware accelerator, which can be written manually or generated through HLS techniques or other means) to the bus interfaces, both for transmitting data and commands to the underlying peripheral using the HW/SW interface and for retrieving the results of the computation through the read interface.

As with those efforts discussed before, *the hardware accelerator also requires of some form of management unit*, typically in the form of a command decoder and state machine for federating the life-cycle of the hardware task, and which strongly resembles the Execution Control Chart of the FB in the IEC 61499 standard. Similarly, external signals and events (i.e. hardware interrupts) can also be used by the IP block through a GPIO interface for interacting with the outside world, much as in the same manner of the process interface described in the IEC 61499 standard.

As it can be inferred from this succinct introduction, many overlapping concerns exist between the aforementioned efforts and those discussed in Section 4.3, but the use of partial reconfiguration has enabled to introduce features currently lacking in those static implementations, namely the reconfiguration and distribution of hardware accelerators into different zones of the FPGA.

The main rationale of PR is the ability to modify areas of the FPGA at run-time, on-demand [55], and in a *quasi glitch-less manner*, as depicted in Fig. 14. For this, the designer needs to specify the zones of the reconfigurable fabric that will be dynamically modified (known as PRRs, for *Partial Reconfigurable Regions*); then, a set of hardware IPs are allocated to these *PRRs* (known as PRMs, for PR Modules). Compared with the traditional approach, various HW tasks can be time-multiplexed for sharing the same physical resources, avoiding many of the pitfalls of the architectures discussed before [46].

Several approaches for defining the reconfigurable areas have been proposed, falling in two main camps: the reconfigurable module can either integrate the bus-wrapped hardware accelerator and the accompanying modules (Fig. 13) or in a second instance, only the hardware accelerator is modified through partial reconfiguration. This choice has some major implications, since the assigned logic blocks are subsequently transformed into *partial bitstreams*, which configure the PRRs on demand at run-time, in order to map the desired functionalities onto the FPGA, as depicted in the figure.

The larger the number of resources encompassing the PRM, the longer the configuration process will take, and issue that can introduce timing penalties and affect the overall system performance. Nonetheless, the added flexibility more than compensates for this drawback in most applications: those architectures described in Sections 4.3.1 and 4.3.2 would require a complete reconfiguration of the FPGA, which will halt the operation of the system during a significantly larger fraction of time, in detriment of the aspects such as the sought availability in DCS platforms, in order to avoid any downtimes or failures.

The reconfiguration process requires some additional components to work. A minimally functional DPR system is encompassed by a *reconfiguration controller (RC)*, in the form of an internal or external processor, to retrieve the generated partial bitstreams from external memory (i.e., Flash or DDR can be used depending on the application) and to drive a special internal port (the ICAP), through which the *reconfiguration process* is carried out. Such minimal system can be used for other purposes than coordinating the partial reconfiguration process, taking more on some managerial role, much in the same manner as the IEC 61499 systems discussed in Section 3.3. Such systems have had successful applications in applications such as software-defined radio [56] and in video processing applications [57], among others.

Despite the strides made in the partial reconfiguration domain, the academic community and many system designers have long sought to develop systems in which the *underlying reconfigurable resources can be treated in a transparent manner*, just as simple threads in a traditional CPU-based design. Such mechanisms could provide a *virtualization layer of the hardware platform* [58], by swapping functionalities in and out of the reconfigurable fabric, at run-time and on-demand. Such features have been considered essential in order *enhance the flexibility of FPGA-based applications*, executing heterogeneous hardware tasks by time-multiplexing the access to predefined physical resources. Additionally, these capabilities could permit the already deployed systems to *adapt to environmental or internal conditions*, by modifying their original configuration to perform a new mission or in self-healing scenarios [59].

Implementing such systems *beyond simple case studies* is however not trivial, as will discuss in the next sub-section. We want to provide the reader with a glimpse of the *complexities associated with the tools*, and the *expertise originally required* for successfully implementing Dynamically

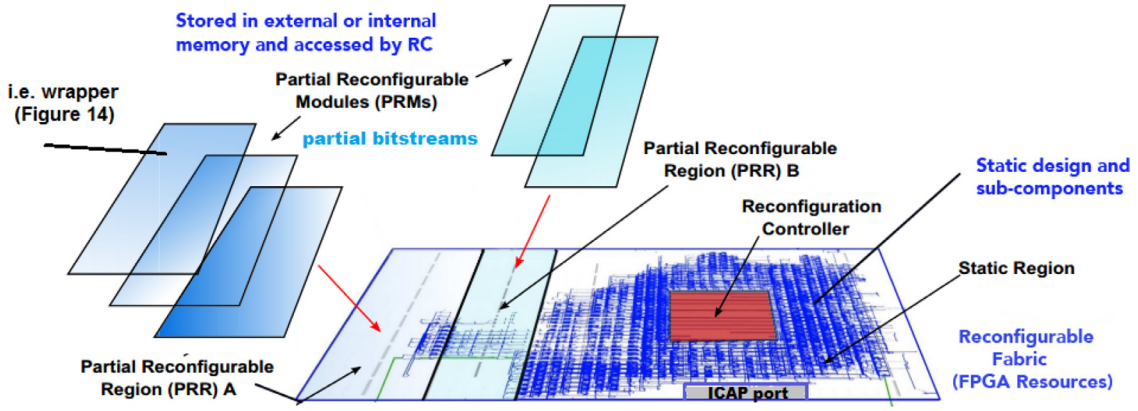

Fig. 14. Dynamic partial reconfiguration basic concepts and process.

Reconfigurable Systems on Chip (DRSoC) using the *traditional DPR design tool flow*. This will provide a framework for the discussion that follows, which intends to demonstrate that new developments in the DPR domain are mature enough to be effectively used in the implementation of the DRSoC architectures necessary for *successfully creating full-fledged IEC 61499 platforms*.

5.2. Using DRSOC for implementing IEC 61499 platforms: advantages and opportunities

As depicted in Fig. 15, a complete IEC 61499 SoC could easily be implemented in an FPGA device. A central processor could cater for the device management, making use of an RTOS executive. A *common virtual bus could be defined and implemented in FPGA*, either via conventional bus protocols or through *tailored PR buses* [60], fostering the development of component-based approaches for IEC 61499 FBs and promoting IP reuse. Moreover, the PR aspects discussed above could aid in the development of full-fledged IEC 61499-based platforms, with increased adaptable heterogeneity.

In the following, we briefly discuss some other potential benefits that could be attained through the deployment of DRSoCs in this regard:

- **Configurability and interoperability**: Using specialized IP wrapping techniques, in tandem with some of the code generation methods discussed in Section 4.3, could lead to better configurability of DRSoC-based IEC 61499 devices, if matched with an holistic model-driven engineering approach [53].

  Such efforts can also have a positive impact in interoperability as well, since the original specification would consist in some of the languages defined by the IEC, and the code generation methods would shield the designer of the technological specific details.
- **Attaining full-fledged reconfiguration and distribution**: Many PLC-like implementations in FPGA devices do not support full-fledged reconfiguration of the control functions implemented in hardware, or distribution of the constituent tasks.

  These issues can be solved by using the architectures and techniques discussed in Section 6.1, to enable full-fledged IEC 61499 DCS platforms, such as bitstream processing and manipulation, known as *bitstream relocation*, which enables to map a hardware task onto different zones of the FPGA [61].
- **Increased performance through FPGA implementation:** Implementing control algorithms in FPGA can significantly improve their performance, as thoroughly discussed in Section 4.3. Such algorithms could be implemented as complex FBs catered by a battery of resource managers (i.e., wrappers) allocated to a reconfigurable regions in the FPGA (i.e.resources), interconnected via a standardized communication interface.

  The algorithms could then be dynamically distributed using a RecOS and the associated services (task relocation and context management, as in our prior work [62]).
- **Encapsulation, reuse and virtualization:** Hardware tasks can coexist with other threads run by the RecOS: they can be swapped in and out of the reconfigurable logic, pre-empted by other processes and reinserted when necessary, or even relocated. The RTE hypervisor caters for the context-management duties, simplifying the

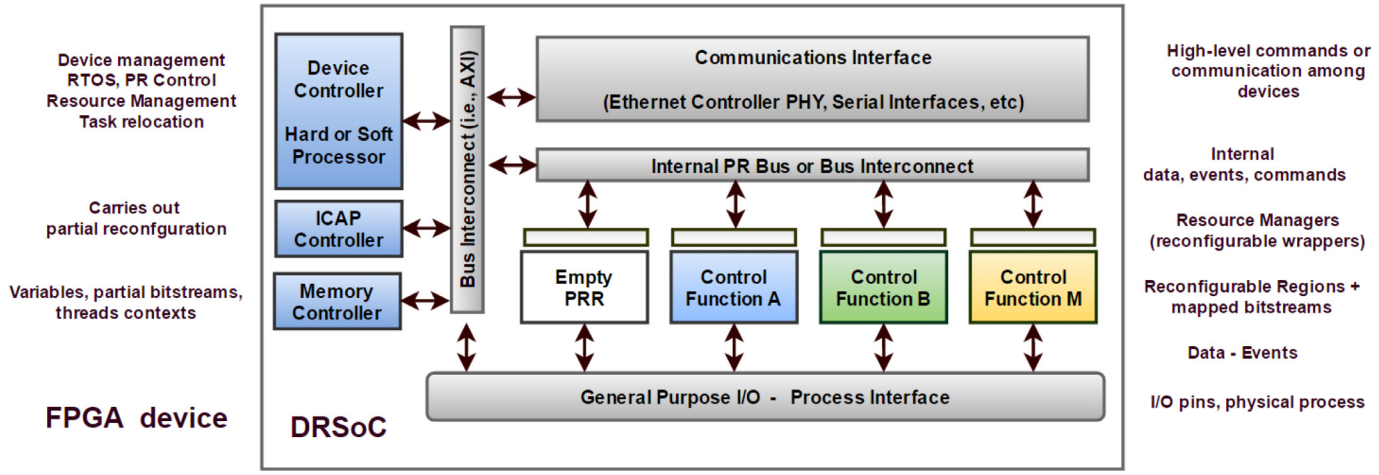

Fig. 15. A conceptual architecture for reconfigurable systems based on DRSoC.

application development and making the entire process transparent to the user, and the specification more maintainable and easy to port and upgrade.

- **Mixing heterogeneous processing policies:** In order to keep pace with the stringent real-time constraints of complex control systems, heterogeneous automation FPGA-based platforms (using either hardened or software implementations of an FB, or a combination of both), could enable extensive design exploration strategies, so the best possible compromise in terms of performance, power consumption, and area/resource utilization, among others policies can be met.
- **Device specialization and support of multiple communication field-bus protocols:** The support for multiple fieldbus protocols in IEDs is an active area of research, but most approaches concentrate on multi-protocol devices, that permit to harmonize the ensemble of the DCS network. The use of partial reconfiguration could permit the creation of customized IEDs, which can be tailored for a given application or network configuration, for supporting a given industrial protocol.
- **Fault-tolerance diagnosis/prognosis:** Monitoring and infotronics mechanisms can be embedded within IEDs, aiding in assessment and prediction of unacceptable behaviors. Such approaches have already been investigated in the reconfigurable computing domain (i.e. hot spot mitigation); moreover, this prognosis approach enables predict and prevent approaches [63].

6. Recent trends in the implementation of DRSoCs

A great deal of research has been made to facilitate the design process of *DRSoC platforms* within the *reconfigurable computing community* [64]. A rather synoptic survey of these efforts will be provided in this section; for a more in-depth discussion, excellent surveys of some of the initial efforts in DPR [65], and of recent more developments [59] are readily available. We will concentrate our discussion on those aspects we deem essential *for building a bridge between the efforts carried within the reconfigurable computing domain, and the opportunities for implementing IEC 61499 systems in reconfigurable devices* discussed in Section 3.4.

First, we will revise some initial efforts to enable the distribution of tasks in DRSoCs, through a process known as *partial bitstream relocation (PBR)*, which can be seen as an extension to the current Xilinx PR design flow. Subsequently, we will discuss how some CAD tools have further extended the capabilities of PR, while reducing the design effort in the process. We posit that such tools could be harnessed to implement the necessary infrastructure to support the reconfiguration and distribution of software and hardware tasks into multiple IEC 61499 devices, overcoming the issues of previous FPGA-based IEC implementations. Afterwards, we will discuss yet another aspect of the design of DRSoC-based platforms, which is concerned with the *management of the reconfiguration process*, which entails not only allocating tasks into predefined physical areas, but *taking into account aspects such as life-cycle of a given task*, the availability of a reconfigurable region, and avoiding any potential hazards due to the reconfiguration process [66].

6.1. Recent developments in tools for partial reconfiguration

In a DRSoC platform, *pre-established portions of a hardware platform* (known as reconfigurable regions) can be changed on-demand, using partial bitstreams (PBs) that map the functionalities of a given *task onto the available resources*. This entails that this configuration data cannot be deployed in other resources within the device and for configuring other devices. This limitation of the traditional design flow has long been considered as a serious obstacle to the implementation of full-fledged reconfigurable systems, since the generated PBs are bounded to such predefined PRRs.

Therefore, the *PB cannot be deployed for configuring reconfigurable logic in a different location*, even if the underlying layouts are identical. This limitation has some significant implications: the required amount memory augments if multiple copies of a task are needed (for instance, for scalable architectures) [61] and furthermore, hampers the relocation or distribution of tasks that is required in many applications (i.e., hot-spot mitigation, fault-tolerant systems).

In order to alleviate some of these issues, and to increase the flexibility of DRSoCs, a method known as *Partial Bitstream Relocation, PBR*) was originally proposed. Most of the works in the literature relied on modifications upon the partial bitstreams obtained from the traditional PR design flow, either by software or by custom hardware accelerators (known as relocators, for instance REPLICA [67]).

A major hurdle in most of these approaches was that, since they were based on the traditional PR design flow, the use of the so-called bus macros was a necessity. This *static interface infrastructure was difficult to use and prone to errors*, with the additional drawback of consuming significant additional resources [59]. This issue was subsequently solved with the introduction of proxy logic interfacing, which in addition to *consuming less resources*, was *introduced automatically* by the tools.

Finally, most PBR approaches supported relatively simple layouts for the PRRs configurations, either as a single or multiple square islands, or *slots in a bus-based architecture*. In many cases, these design choices stemmed from limitations in the deployed place and route strategies, which did not enable *more complex geometries and interconnection infrastructures*; a good overview of PBR can be found in previous works by Touiza and Ochoa-Ruiz [61]. Nonetheless, as we will see, these issues have been *largely tackled by new tools and methods*, paving the way to more sophisticated DRSoC platforms.

The issues mentioned above have been recognized by the community at large, which have made strides in the development of design *tools to facilitate the design process of PBR-capable reconfigurable systems*. In order to circumvent the above mentioned issues, such design tools needed to *access low-level FPGA primitives* such as LUTs, FFs, and switch matrix multiplexers [59], which cannot be directly manipulated using hardware description languages (HDLs).

Therefore, several design tools, such as RapidSmith [64], exploited intermediate design descriptions, such as the *Xilinx Description Language* (XDL), which offers a readable and manipulable description of low-level netlists, enabling the designer a greater control over the mapping and place and route stages of the FPGA design process. This *increased control on the placement and routing of reconfigurable modules* has the additional benefit of creating more resource-efficient DRSoCs [62].

A great deal of research and tool development, based on the original propositions of RapidSmith has been carried out, with increased levels of sophistication and easy-of-use. Due to space considerations, we will just briefly describe some of the most prominent features of such tools, summarized in Fig. 16, to showcase the progresses made in the domain; for a more detailed account of the evolution of the such tools, excellent sources exist [68].

Some initial endeavors were represented by the DAPR [69] and OpenPR [70] design flows. Although interesting in their inception, they had several issues, which stemmed mainly from their use of bus macros for interfacing the RMs with the static design. This characteristic severely limited their use in PBR-capable designs, since the former does not support relocation (it is basically an extension to the capabilities of PlanAhead), whereas the latter supports distribution of tasks only between PRR with identical layouts. Moreover, the implementation of PR platforms with such tools *was deemed very time-consuming*.

More recent tools have circumvented various drawbacks of the approaches described above. For one, tools such as Dreams [71] and OORBIT [61] provided *streamlined methods for floor-planning the PRRs*, independent from the cumbersome front-end provided by PlanAhead.

Additionally, these tools avoided resource consuming interfacing methods such as bus macros, favoring proxy logic or partition pins,

Tool/Features	PlanAhead	OpenPR	DREAMS	OORBIT	GoAhead	RePaBit
Reference	[Xilinx, 2015]	[Athanas, 2011]	[Otero, 2012]	[Ochoa-Ruiz, 2013]	[Bechkoff, 2014]	[Rettkowski, 2018]
Design front-end	No	No	No	No	Yes	Yes
Floorplanning	Manual + GUI	–	Manual + GUI	Automated	Automated	Automated
Interface method	BM and PL	BM	PL	PL	PL	BM - PL -ZL
Module Relocation	No	Yes	Yes	Yes	Yes	Yes
Type of Relocation	N/A	Same layout	Same layout	Same layout	Fine grained	Fine grained
PRR layout type	SI	SI, MI, Slot	All	All	All	All

Fig. 16. Comparison of various extended DPR design tools adapted and extended from [59].

which are more *latency-efficient and are automatically inferred and inserted*.

These approaches enable as well complex PRR geometries (slot and grid-based systems are possible), albeit with the limitation of restricting the PRRs identical shapes and underlying resources (i.e., same layout). Furthermore, they suffer of *relatively long compilation times* (for the synthesis of the module netlists, a curse in most FPGA-based design tools), exacerbated by the time overhead incurred by the generation of the PR static and partial bitstreams, which stems from the *multiples passages through the design flow to avoid any routing conflicts* among the various static and partial bitstream configurations.

Many of these issues were solved by subsequent tools, in particular GoAhead [59], which couples a floor-planning front-end with a battery of automated scripts. Some features of the tool were the generation of constraints for the underlying implementation tools (i.e., mapping, P&R), HDL code generation (of templates and interfaces) and support for various static-PRR interfaces choices (BM, PL and the so-called zero logic, which incurs in very low resource overhead).

Furthermore, various PRR layout types and relocation methods were supported. These capabilities are possible due to the *communication infrastructure* upon which GoAhead is based, ReCoBus (a homogeneously structured hard macro) [60]. This infrastructure enables a very fine-grained placement of modules, and *imposes less restrictions on the relocation of tasks*, which are not limited to the size of the PRR nor the underlying physical resources. This approach has an additional benefit: the PRRs do not need to be over-specified, a major hurdle in the vast majority of approaches to DPR design [72].

Despite the progresses made by the reconfigurable computing community working in PR, with the introduction of the tool Vivado by Xilinx in 2013 [73], many of the progresses described above have been rendered obsolete or have forced the research community to continue using old tools and devices. This is due to the fact that Xilinx introduced a number of changes to its CAD tool, including the discontinuance of XDL; without XDL, the research tools discussed above are rendered virtually incompatible with future generation of reconfigurable devices. Instead, Vivado provides access to its design and device databases through the generation of external EDIF and Xilinx Design Constraints (XDC) files via a Tcl interface.

This Tcl interface now provides means to access key functionalities required to create, place, and route modules and designs, much in the same way XDL did, without relying explicitly on PlanAhead, which further abstracts the PR design process. Recent research efforts such as the work of Oomen et al. [74] have exploited this interface to create tools to enable the automated generation of rellocatable partial bitstreams, which require access to the placement of various primitives to avoid hazards during the relocation process. More recently, tools as TINCR [75] and RePaBit [76] have been proposed, rendering the design process even more amenable for the user; it must be noted the second makes use of the Xilinx Isolation Design Flow [77] to enable the design of full-fledged DPR systems using the Xilinx Vivado PR Toolchain [78]. Finally, it is worth noting that RePaBit supports many new devices not targeted by other CAD tools, such as the All-Programmable Zynq family of devices, which we believe is an excellent candidate for implementing IEC 61499 enabled reconfigurable platforms as the one outlined in 5.2 and depicted in Fig. 15.

All the tools described above have *fostered the use of ESL techniques for the creation of DRSoC* [53], raising the level of abstraction. Furthermore, they have continually made the *reconfiguration process more flexible and transparent*, by shielding the budgeting of resources and relocation of tasks from the designer. As we will see in the next sections, the relocation capabilities provided by these tools is essential to create Dynamically Reconfigurable Systems on Chip, if paired with other developments made in parallel by the reconfigurable computing communit: namely, preemptable hardware modules and the accompanying Reconfigurable OS to underpin it all.

6.2. Reconfiguration management and tasks relocation and reconfiguration in DRSoCs

The strides made in CAD tools for PR described above have unleashed many possibilities in system design, since they have enabled researchers to express and conceptualize on-demand hardware adaptability in a more high-level manner, without the burden of being preoccupied with low-level implementation aspects inherent to the technology. Nonetheless, a PR platform only becomes truly a dynamically reconfigurable SoC (DRSoC) through the use of an embedded reconfiguration management executive and a sufficient level of virtualization.

The latter is required to in order to make the management of the PRMs and their allocation onto the reconfigurable resources less daunting from an application development perspective [79]. In this manner, the PRMs are no longer merely static circuits performing a given function, but become resources that can be exploited by the application (along traditional software functions) through a simplified programming model [80].

In order to accomplish these lofty goals, the system management executive needs to be conceived in a manner that goes *beyond the relatively simple reconfiguration controller* outlined in Section 5.1, providing more OS-like capabilities to support features such as *task preemption, context management, relocation, and scheduling*. In contrast to software-based OS, the context of the hardware threads is not only stored in a memory stack, but some of the internal variables are located within logic resources used to implement the module [81], and needs to be taken care of if the task is to be preempted by another thread with a higher priority. This process, managed by an augmented reconfiguration controller, is to be analyzed in the next section.

6.2.1. Reconfiguration management and control of dynamically partial reconfigurable systems

A number of research efforts have been undertaken during the last decade to build more *flexible DRSoCs*, with especial emphasis on the proposal of novel *reconfigurable OSes* (*RecOS* henceforth) underpinning it all. The term of reconfigurable OS essentially refers to an hypervisor augmented with services to manage the underlying reconfigurable areas and execute functions on it [82], providing means for the management of software and hardware tasks, as well as the *access to the underlying*

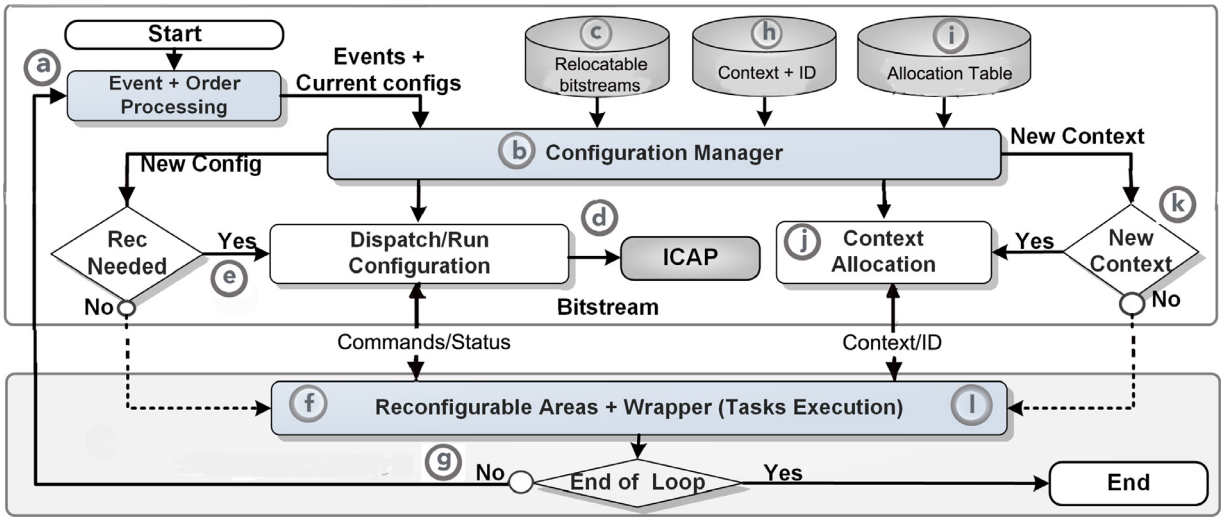

Fig. 17. Reconfiguration management for DRSoCs in tandem with relocation and context-management.

hardware resources (i.e., memory, PRRs, peripherals).

The execution of the application and the implementation of the partial reconfiguration process can follow different policies, enforced by the so-called *Reconfiguration controller*, as depicted in Fig. 17 above.

Typically, the internal system configuration can be modified (at runtime) depending on the arrival of external events ⓐ (i.e. a change of format in a video chain), internal application dynamics (a task is to be preempted by another) or for instance, to recover from unexpected faults. In the same manner, the RC can also *map an empty bitstream* whenever a function is *no longer needed*, effectively releasing the occupied resources and fostering power management strategies.

In basic PR systems, this *configuration management function* ⓑ can be as simple as a *state machine* written by hand or a relatively complex management software running on a processor. As discussed in Section 5.1, the basic role of the RC is *dispatching the partial bitstream (PB)* for mapping the new functionality ⓒ. This entails taking control of the ICAP controller, transmitting the reconfiguration data for a given reconfigurable area, and retrieving the PBs from external memory ⓓ.

In the vast majority of traditional DRSoC designs, the reconfiguration controllers were only capable of mapping of non-relocatable bitstreams onto predefined reconfigurable regions. The main role of the RC was to coordinate the interaction between the intended application and catering the correct reconfiguration management [79], letting the tasks execution undisturbed if changes were not required ⓔ. Conversely, a new task can be allocated a predefined region ⓕ when necessary, until the completion of an application loop ⓖ.

This process was carried out without taking into account the internal state of the task, this is, even if some thread was running before preempting it. This is due to the fact that was usually assumed that a task was not to be resumed, and this its internal context and location within the fabric were not taken care of. Such an approach severely limits the kind of applications that can be tackled with DRSoCs and thus may endeavors have introduced *context management capabilities* into the RC to support OS-like management of reconfigurable hardware tasks.

Nowadays, the vast majority of RecOSes provide *software APIs for handling the hardware and software tasks* and support relocation of the former, using existing mechanisms in standard RTOSes [62]. The rational for this approach is to take advantage of the scheduling algorithms integrated into the RTOSes, with the additional benefits of exploiting the HW/SW interface as a means of communication with the running processes, as well as fostering IP reuse through the use of bus wrappers, as depicted in Fig. 18.

In order to support context-management capabilities, the RC needs to be able to interact with the underlying hardware modules, in order to *monitor their status* (i.e. whether or not it *can be preempted*), as well as to control their life-cycle, through commands issued by the RecOS. Moreover, the RC also needs to be able to *inject and retrieve the context* ⓗ of the tasks during a removal or insertion operation (depending on the availability of PRRs in the reconfigurable zone of the FPGA ⓘ), an operation carried out by the context allocation manager ⓙ. The communication with the hardware tasks in achieved though the a virtualization layer provided by the HW/SW interface of specialized *context-management wrappers*, which serve as the gateway between the high-level RecOS and the reconfigurable regions and the allocated tasks.

As PBs can be relocated to any available PRR, the tasks must be

Metric / Approach (1)	Replica	ReconOS	FUSE	RAMPSoCVM	HTR
Reference	[Kalte, 2005]	[Lübbers, 2009]	[Ismail, 2011]	[Göhringer, 2011]	[M-Villanueva, 2013]
Type Wrapper/ Bus IF	PLB + Custom	PLB + OSIF	PLB + Custom	Custom	Custom
RecOS API Support	No	Yes	Yes	Yes	Yes
Preepmption and Context Mgmnt	No	Yes	Yes	Yes	Yes

Metric / Approach (2)	Dyn. Objects	R3TOS	Recoblocks	CODEZERO	FAMOUS
Reference	[Dondo, 2013]	[Iturbe, 2013]	[Navas, 2013]	[Jain, 2014]	[Ochoa-Ruiz, 2018]
Type Wrapper/ Bus IF	PLB + Custom	Custom	AXI + Custom	AXI + Custom	AXI + Custom
RecOS API Support	Yes	Yes	Yes	Yes	Yes
Preepmption and Context Mgmnt	Yes	Yes	Yes	Yes	Yes

Fig. 18. Comparison of reconfiguration management and RecOes, extended from [62].

correctly identified as to avoid any errors during the re-injection of a task with a given context (k). Hence, the RC requires an *increased awareness of the state of the reconfigurable section* of the FPGA (in terms of which areas are occupied or not, and with which contexts, which is aided by the use of an allocation table (k)). These elements help the RC to manage the complexity of the underlying hardware resources, allocating only a new context when necessary (m), or otherwise inserting the original context along the correct PB.

Several RecOSes implementations have been proposed in recent years, such as OS4RS [81], ReconOS [80], RAMPSoCVM [83], and more recently R3TOS [82] and CODEZERO [84]. A good overview of this area can be found in the last two references (summarized in Fig. 18). These approaches have progressively enabled, due to advances in PR tools, an *increasing sophistication in the management of full-fledged DRSoCs*, implementing many of the capabilities and characteristics sought by the device management provisions of the IEC 61499 standard (Fig. 6, Section 3.3).

These developments are very much aligned with what some *researchers within the automation community* have proposed in recent years: using RTOSes as a means to implement the run-time environment (RTE) for IEC 61499-based devices [85]. This research points at the need of *virtualizing the communication between the RTE and the various resources in a device*, so that the control engineers can *develop applications for a neutral platform*, leaving the low-level implementation details to designers of the function blocks.

In order to fully incorporate hardware virtualization in DRSoCs, the *use of RecOSes alone is not enough*, since the entire endeavor has to be underpinned by *preemptable and relocatable hardware modules* (Section 6.1). We have already discussed how relocatable modules can be created, using either bitstream modification means or more advanced tools for creation DRSoCs (Section 6.2).

The context management features, as typically supported by today's RecOes will be discussed in the next section, which deals with how the preemptable HW modules are designed.

6.2.2. Component-based framework context and relocation management

Supporting *thread-like features to hardware tasks* entails conceiving the reconfigurable modules (RMs) as fully independent objects, with consistent interfaces and a *set of APIs for effective task preemption* strategies. Similar to *software tasks*, reconfigurable *hardware modules* in DRSoC platforms also need to be managed and scheduled, which entails an awareness of the modules' placements and current contexts. Hardware threads are no longer merely passive components attached to the bus and controlled though a software API: they can be reactive and contain internal *state* and *context* information.

The mechanisms necessary to *guarantee the consistence and coherence of the relocated and/or preempted tasks* is known as a *context management* [86]. Therefore, in recent years, many works have pointed towards the specification of hardware components as tasks, from an *object-oriented perspective*: RMs need to be designed in such a way that they can be *safely stopped* via *checkpoints* and a task management unit (*i.e. context retrieval and reinjection, relocation*) must be implemented both at the component and RecOS level.

Therefore, *abstraction mechanisms* must be provided so that the low-level implementation details associated with the HW components are *transparent from the application level perspective*. Some examples of such wrappers have been presented in the last few years [87], with varying levels of complexity [51].

However, the majority of approaches share many commonalities, regarding the manner in which these components interact with the RecOS *as a bridge to the modules to be executed in the PRR* [86]. An example of such a wrapper [88] (proposed by the authors), and the associated process state diagram (PSD) for a task are shown on Fig. 19 (a) and (b), respectively.

The rationale behind this wrapping is that the task management services can be implemented in hardware, *discharging the processor of these time-consuming duties*. Thus, the wrappers serve as a bridge between the complex bus protocol signaling and the actual sub-module functionalities (i.e., I/Os, hardware accelerators).

In this sense, it is very *akin to the resource implementation of the IEC 61499 standard*, enabling the modification and *relocation of functions (function blocks) onto resources (PRRs)*.

However, the most important feature of these modules is that they *exploit object-oriented strategies to achieve higher degrees of HW virtualization*. In this manner, the wrapper enables to cater for the management the state and context of the associated RM, *in a portable and data efficient manner*.

This is typically attained through the implementation of a set of services at the RecOS level, which translate into specific actions at the hardware level, for carrying out actions such as *creating a thread* (i.e., create task, set context) or *removing it from the PRR* (stop, disabling and killing a task), which entail the manipulation and relocation of the task bitstream.

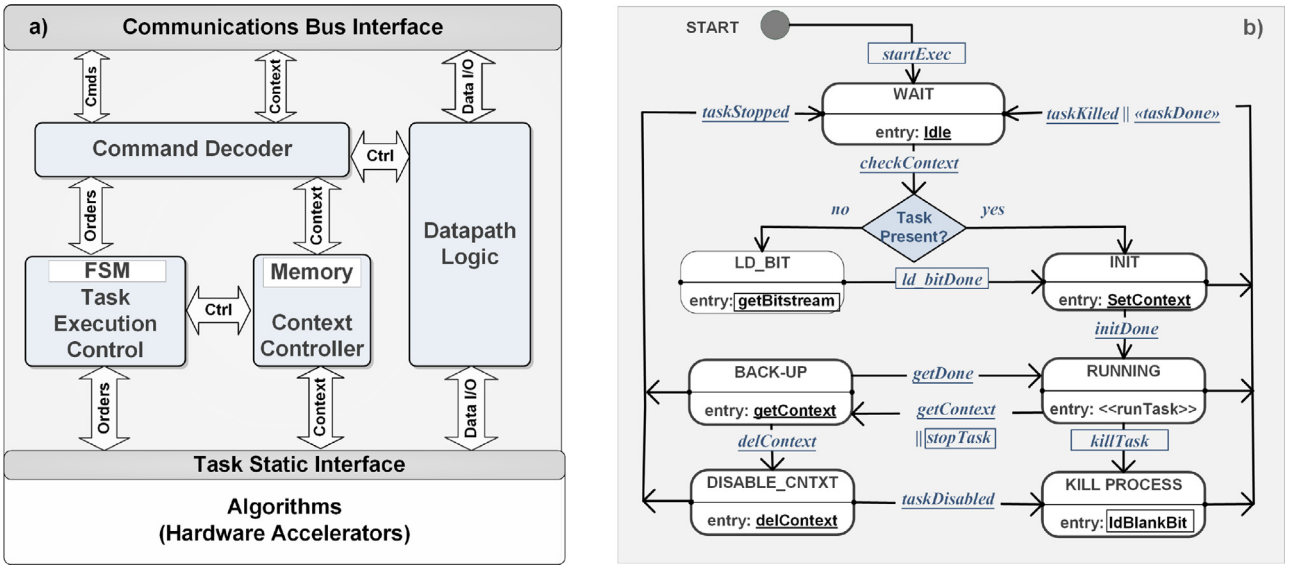

Fig. 19. (a) A typical wrapper for context management (b) Internal FSM for interacting with the PR executive.

These activities are typically coordinated by an internal control unit (a command decoder in Fig. 19 (a)) which serves as the interface between the RC management and the wrapper, while *federating the lifecycle of the allocated task* through an FSM (the PSD in Fig. 19(b)). The FSM takes into the account the state of the task before inserting or removing it from the PRR (*effectively enabling the required checkpoints for the application*) and informing a *context controller that data is to be temporally stored*, either internally in the wrapper or in external memory [62].

As can be inferred from this brief introduction to the reconfiguration management of DRSoCs, many of the endeavors carried out by the reconfigurable computing community in partially reconfigurable systems are not very far from the ongoing research on the IEC 61499 standard.

Specifically, the device management specification aspects of the IEC 61499 standard, discussed in Section 3.3 have been practically implemented in DRSoC platforms, with the distribution and reconfiguration capabilities already in place thanks to the progresses made in partial reconfiguration technology and tools (such as bitstream relocation).

The manner in which the resource and function block models have been implemented in some of the works discussed in Section 4.3 point at many commonalities as well [46], with the FB implementation of Fig. 11 being strikingly similar to the wrapper implementations introduced above. The same applies for the process state diagram (PSD) used to model the behavior of a task in many current DRSoC implementations, which practically implements the PSD described in the IEC 61499 (Fig. 6), albeit in a massively parallel architecture setting.

However, despite of the increased sophistication and versatility in system conception which can be attained by using DRSoCs, a more holistic approach is needed in order to exploit such kind of platforms for implementing heterogeneous IEC 61499 applications. Specifically, the code generation approaches discussed in Section 4.3 need to be accompanied by interface synthesis and automatic integration in predefined and widely accepted bus-wrappers, either as dynamic objects with internal context or simple hardware threads that can be relocated onto various resources, either as software and hardware implementations (i.e. the Hardware Function Blocks in [47]). In this manner, the design process becomes more transparent from the application development perspective [53].

The platform description and the behavior of the DRSoC-based applications also needs to be fully accessible by these code generation tools, in order to incorporate the context management and relocation capabilities. A great deal of research in this regard has been carried out by the model-driven engineering community, through the introduction of several design tools for describing and implementing Systems-On-Chip and subsequently, DRSoCs . In particular, frameworks for managing such types of problems in an holistic manner have been proposed over the years by the MDE community, but more efforts need to be done to close the gap between the two domains and to be able to implement full-fledged IEC 61499 systems using DRSoCs [62].

6.3. Challenges for the convergence of DRSoC paradigm and the IEC 61499 standard

In the previous sections, we have delved into an in-depth survey of the strides made by the reconfigurable computing (RC) community to advance the partial reconfiguration technology. These progresses have enabled system designers to move from relatively simple, *static partitioning strategies* (in which independent task could simply be swapped in an out of the reconfigurable fabric), to more malleable platforms, which we have dubbed *Dynamically Reconfigurable Systems-on-Chip*. The main goal of this article was to highlight the many resemblances between many of the efforts carried out by the RC community in recent years, and the previsions of the IEC 61499 norm, in terms of how the devices based on the norm are expected to work.

Furthermore, we have provided some convincing arguments about what we believe is currently lacking in both software and hardware implementations of the IEC norms (i.e. MCU and FPGA-based implementations, respectively) and how many of these shortcomings could be circumvented through the use of advanced partial reconfiguration tools and techniques, thoroughly discussed in this section. We have focused our discussion mainly on the architectural aspects of the norm, and *drawn the many parallels that exist between the device manager and resource models in the IEC 61499 norm*, and many of the implementations currently realized by the academia in the reconfigurable computing field, specifically in dynamically reconfigurable architectures.

Nonetheless, the research in reconfigurable computing has not been limited to the architectural aspects discussed in the previous sections. These developments have just enabled the realization of full-fledged dynamic hardware objects or swappable (and preemptable) logic units [58] (the context-aware wrappers discussed in Section 6.2), which have permitted to implement many of the features currently lacking in the FPGA-based implementations discussed in Section 4.3, namely: improving the configurability of FPGA-based applications, enabling fine-grained reconfiguration of HW tasks in many applications and finally, the distribution (i.e., relocation) of hardware threads within resources (i.e., reconfigurable regions) in DRSoCs, which have been exploited in a large number of applications.

Additionally, these capabilities enable the creation of autonomous or self-reliant platforms, capable of adapting to changes on the environment or of implementing self-healing capabilities, using the very same partial bitstream relocation methods we have discussed above [89]. For instance, a great deal of research has been conducted in the areas of hot-spot mitigation [90] and in fault-tolerant systems [91], in which the reconfiguration controller or RecOS detects an unexpected behavior and can map the associated functionality to other section of the FPGA through relocation, without halting the device and therefore, the entire application that depends on the reconfigurable device or submodule. This of course fall under the umbrella of the much required availability of DCS systems, which is especially essential in industrial and manufacturing settings.

Other features related to partial reconfiguration not addressed in this article pertain to aspects such as HW/SW co-design, the partitioning of applications into software and hardware functions for design space exploration purposes and finally, the best scheduling strategies to be deployed within a particular application [92]. These are important issues, since the partial reconfiguration process incurs in latency penalties that need to be taken into consideration during profiling of the application for a given DRSoC platform [93], in order to obtain the best compromise between performance and power consumption.

A great deal of research has been carried out using static scheduling strategies [79] (the behavior of the DRSoC and PRRs is predefined offline), but with the introduction of context-aware and preemptable hardware threads, as well as some of the RecOSes briefly discussed in Section 6.2, the possibilities for full-fledged dynamic scheduling, as those required by the IEC 61499 are not that far ahead [62]. However, we consider that a discussion on such topics is out of the scope of this article and could be better served if addressed in a separate comparative survey, but excellent sources already exist [94].

We believe that the progresses discussed in this section could have a positive impact in overcoming the limitations currently associated with IEC 61499 hardware implementations. The virtualization capabilities provided by DRSoCs could foster research in the automation community and enable capabilities such as distribution and reconfiguration, currently lacking in mostly static implementations of the standard. Moreover, they could enable to seamlessly integrate and explore various scheduling and device management policies using a common architectural framework and potentially further promote IP exchange among the various actors in the automation community. Furthermore, several interesting developments in the High-level synthesis could provide the additional boost required to bridge the gap that has

traditionally hampered the use of reconfigurable devices beyond specialized application areas [40].

However, the challenges for the convergence of these various disparate concerns demand innovative modeling tools. A lot of recent research suggests that one of the most promising routes is to make extensive use of MDE techniques, accompanied by the encapsulation of the various control algorithms into multi-language and multi-platform libraries [20]. More importantly, the incorporation of adequate architecture models for several target technologies, in order to foster interoperability is a necessity, but adequate models of execution and computation are also required in order to tame the complexity of the DCS in which such devices are to be integrated.

Furthermore, a major challenge in the adoption of new technological paradigms comes from the engineering perspective: in order to be used by the industry, the perceived migration cost associated with new tools and technologies should be significantly smaller than the perceived benefit. In order to overcome the much maligned learning curves and migration costs, the entire endeavor needs to be underpinned as well by appropriate tools, so the various experts can concentrate in describing their respective concerns, letting the tools to cater for the implementation details, though automated procedures, which could be solved through the use of the MDE methodologies.

7. Conclusions and final remarks

In this paper, we have discussed the current state of the IEC 61499 standard, which was introduced as a means to architect multi-platform DCS from a common application model. The provisions of the IEC for this one and other standards have proven insufficient to attain the much desired features of agile systems. The first two requirements (portability and configurability) have been partially solved via the standardization and programming languages, while the third (interoperability) still necessitates the successful incorporation of other standards, such as the IEC 64424, in combination with a Model-driven Engineering approach.

Nonetheless, the main of issues hampering the adoption of the IEC 61499 are not only methodological, but stem in no small part from the lack of common and well accepted reference hardware models. We have briefly discussed how SoC FPGAs could be used for successfully implementing IEC 61499-based systems, with several advantages such the capability to explore various device management strategies and resource allocation of HW or SW tasks and the possibility to implement full-fledged reconfiguration and distribution.

In particular, we have discussed several progresses made by the reconfigurable computing community, which are strikingly close to those sought by the IEC 61499 standard and which have not been addressed by the ongoing efforts in the implementation of the standard in reconfigurable devices.

We believe that such progresses, pertaining new capabilities in the creation of DRSoCs (such as task management and relocation in tile- and slot-based platforms) could have a positive impact in overcoming the limitations currently associated with IEC 61499 hardware implementations. The virtualization capabilities provided by dynamically reconfigurable SoC-FPGAs could foster research in the automation community and enable capabilities such as distribution and reconfiguration, currently lacking in mostly static implementations of the standard.

Moreover, they could enable to seamlessly integrate and explore various scheduling and device management policies using a common architectural framework and potentially further promote IP exchange among the various actors in the automation community. Furthermore, several interesting developments in the High-level synthesis could provide the additional boost required to bridge the gap that has traditionally hampered the use of reconfigurable devices beyond specialized application areas.

However, as only hinted at the end of the article, the challenges for the convergence of these various disparate concerns demand innovative modeling paradigms. A lot of recent research suggests that one of the most promising routes is to make extensive use of Model-driven Engineering techniques, accompanied by the encapsulation of the various control algorithms (i.e., Function Blocks) into multi-language and multi-platform libraries. More importantly, the incorporation of adequate architecture models for several target technologies, in order to foster interoperability is a necessity, but adequate models of execution and computation are also required in order to tame the complexity of the DCS in which such devices are to be integrated.

These topics have been an active area of research in the last several years, but most frameworks do not take into account FPGAs as a candidate for implementing IEC 61499 systems, concentrating more in microcontroller or micro-processor-based systems. As we discussed in the article, some very incipient implementations of the standard in FPGAs have started to emerge, but they do not support the heterogeneity and reconfiguration capabilities necessary to implement usable and fullfledged IEC 61499 devices. For this, more advanced design techniques and tools need to be deployed, as described in this paper: namely, partial reconfiguration in tandem with DRSoCs running RecOSes customly designed for implementing the provisions of the IEC 61499 standard.

Nonetheless, an important lesson that can be gained from the study of the evolution in CAD tools for creating partially reconfigurable systems is that there exists a divide between what the academia can make in terms of research and what the tools vendors offer to their core clients and so, keeping the pace with each new development is also a major challenge for creating innovative CAD tools for niche markets and applications, as is the case for the partial reconfiguration technology [95,96]. However, as we have seen, the potential for PR as a disruptor in many fields has been proved again and again, and we believe that the implementation of the IEC 61499-compliant FPGA-based architectures is a field worth of further research.

References

[1] C. Gerber, H.-M. Hanisch, S. Ebbinghaus, From iec 61131 to iec 61499 for distributed systems: a case study, EURASIP J. Embedded Syst. 2008 (2008) 4:1–4:8.
[2] K. Thramboulidis, Challenges in the development of mechatronic systems: the mechatronic component, Emerging Technologies and Factory Automation. IEEE International Conference on, (2008), pp. 624–631.
[3] C. Sunder, A. Zoitl, J. Christensen, H. Steininger, J. Rritsche, Considering iec 61131-3 and iec 61499 in the context of component frameworks, Ind. Informatics, 6th IEEE Int. Conf. on, (2008), pp. 277–282.
[4] F. Andren, T. Strasser, A. Zoitl, I. Hegny, A reconfigurable communication gateway for distributed embedded control systems, 38th Annual Conference on IEEE Industrial Electronics Society, (2012).
[5] S. Patil, J. Yan, V. Vyatkin, C. Pang, On composition of mechatronic components enabled by interoperability and portability provisions of iec 61499: A case study, Emerging Technologies Factory Automation (ETFA), 2013 IEEE 18th Conference on, (2013), pp. 1–4.
[6] A. Astarloa, Intelligent gateways make a factory smarter, Xilinx Xcell Journal Vol. 94, (2016), pp. 14–21.
[7] IEC, International Standard IEC 6113-3: Part 3: Programming Languages, IEC, 1993.
[8] V. Vyatkin, J. Christensen, J. Lastra, F. Auinger, Oooneida: an open, object-oriented knowledge economy for intelligent distributed automation, Industrial Informatics, 2003. INDIN 2003. Proceedings. IEEE International Conference on, (2003), pp. 79–88.
[9] A. Zoitl, T. Strasser, C. Sunder, T. Baier, Is iec 61499 in harmony with iec 61131-3? Ind. Electron. Mag. IEEE 3 (4) (2009) 49–55.
[10] IEC, International Standard IEC 61499-1: Function Blocks - Part 1: Architecture, IEC, 2005.
[11] R. Lewis, Modeling Control Systems using IEC 61499, IEE, 2001.
[12] V. Dubinin, V. Vyatkin, On definition of a formal model for iec 61499 function blocks, EURASIP J. Embedded Syst. 2008 (1) (2008) 426713.
[13] D. Wenbin, V. Vyatkin, Redesign distributed plc control systems using iec 61499 function blocks, Autom. Sci. Eng. IEEE Trans. 9 (2) (2012) 390–401.
[14] OMG, Model-driven engineering, 2014, http://www.ogm/org/mda.
[15] T. Strasser, A. Zoitl, J. Christensen, C. Sunder, Design and execution issues in iec 61499 distributed automation and control systems, Syst. Man Cybern. Part C Appl. Rev. IEEE Trans. 41 (1) (2011) 41–51.
[16] V. Vyatkin, The iec 61499 standard and its semantics, Ind. Electron. Mag. IEEE 3 (4) (2009) 40–48.
[17] A. Zoitl, Real-Time Execution for IEC 61499, ISA, 2009.
[18] D. O'Sullivan, D. Heffernan, Vhdl architecture for iec 61499 function blocks, Comput. Digital Tech. IET 4 (6) (2010) 515–524.

[19] E. Monmasson, L. Idkhajine, M.W. Naouar, Fpga-based controllers, Ind. Electron. Mag. IEEE 5 (1) (2011) 14–26.
[20] V. Vyatkin, Software engineering in industrial automation: state-of-the-art review, Ind. Inf. IEEE Trans. 9 (3) (2013) 1234–1249.
[21] W. Dai, V. Vyatkin, J. Christensen, Essential elements for programming of distributed automation and control systems, Emerging Technologies Factory Automation (ETFA), 2013 IEEE 18th Conference on, (2013), pp. 1–8.
[22] K. Thramboulidis, Iec 61499: Back to the well proven practice of iec 61131? Emerging Technologies Factory Automation (ETFA), 2012 IEEE 17th Conference on, (2012), pp. 1–8.
[23] I. HOLOBLOC, Fbdk 2.6 - the function block development kit, 2018, http://www.holobloc.com/fbdk2/index.htm.
[24] A. Zoitl, T. Strasser, G. Ebenhofer, Developing modular reusable iec 61499 control applications with 4diac, Industrial Informatics (INDIN), 2013 11th IEEE International Conference on, (2013), pp. 358–363.
[25] R. Automation, Isagraf: Software for next generation automation and control, 2018, http://www.isagraf.com/get/oct06pdfs/SW-nextgen-automation-and-control.pdf.
[26] J. Goeders, G. Holland, L. Shannon, S. Wilton, Chapter 14: systems-on-chip on fpgas, in: D. Koch, F. Hannig, D. Ziener (Eds.), FPGAs for Software Programmers, Springer Netherlands, 2015. 461–283
[27] R. Hartenstein, Chapter 20: basics of reconfigurable computing, 2007.
[28] M. Santarini, Xilinx fpgas: Creating a greener future for industrial motor control, Xilinx Xcell Journal Vol. 73, Fourh Quqrter, (2010), pp. 8–11.
[29] L. Gomes, E. Monmasson, M. Cirstea, J. Rodriguez-Andina, Industrial electronic control: Fpgas and embedded systems solutions, Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE, (2013), pp. 60–65.
[30] M. Chmiel, J. Mocha, E. Hrynkiewicz, A fpga-based bit-word {PLC} {CPUs} development platform, {IFAC} Proc. Vol. 43 (24) (2010) 132–137. 10th {IFAC} Workshop on Programmable Devices and Embedded Systems.
[31] A. Chang, Innovative platform-based design for the industrial internet of things, Xilinx Xcell Journal Vol. 92, (2015), pp. 32–37.
[32] I. Kuon, R. Rose, Quantifying and Exploring the Gap Between FPGAs and ASICs, Springer, 2010.
[33] R. Hartenstein, T. Kaiserslautern, Chapter 22: applications, design tools and low power issues in fpga reconfiguration, in: J. Henkel, S. Parameswaran (Eds.), Designing Embedded Processors, Springer Netherlands, 2007, pp. 451–501, , https://doi.org/10.1007/978-1-4020-5869-1_20.
[34] C. Maxfield, FPGA:s World Class Designs, 1st, Newnes, 2009.
[35] R. Dubey, Embedded System Design Using Field Programmable Gate Arrays, 1st, Springer, 2009.
[36] R. Woods, J. McAllister, G. Lightbody, Y. Yi, FPGA-based Implementation of Signal Processing Systems, 1st, Wiley, 2008.
[37] D. Blouin, G. Ochoa-Ruiz, Y. Eustache, J.P. Diguet, Kaolin: a system-level aadl tool for fpga design reuse, upgrade and migration, Adaptive Hardware and Systems (AHS), 2015 NASA/ESA Conference on, (2015), pp. 1–8.
[38] D. Densmore, R. Passerone, A platform-based taxonomy for esl design, IEEE Des. Test Comput. 23 (5) (2006) 359–374, https://doi.org/10.1109/MDT.2006.112.
[39] P. Coussy, High-Level Synthesis: From Algorithm to Digital Circuit, Springer, 2008.
[40] A. Sangiovanni-Vincentelli, Quo vadis, sld? reasoning about the trends and challenges of system level design, Proc. IEEE 95 (3) (2007) 467–506.
[41] P. Coussy, D.D. Gajski, M. Meredith, A. Takach, An introduction to high-level synthesis, IEEE Des. Test Comput. 26 (4) (2009) 8–17, https://doi.org/10.1109/MDT.2009.69.
[42] A. Gerstlauer, C. Haubelt, A.D. Pimentel, T.P. Stefanov, D.D. Gajski, J. Teich, Electronic system-level synthesis methodologies, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 28 (10) (2009) 1517–1530, https://doi.org/10.1109/TCAD.2009.2026356.
[43] A. Milik, On hardware synthesis and implementation of {PLC} programs in {FPGAs}, Microprocess. Microsyst. 44 (2016) 2–16. Modern Techniques of Design and Implementation of Highly Flexible Controllers.
[44] D. Du, X. Xu, K. Yamazaki, A study on the generation of silicon-based hardware plc by means of the direct conversion of the ladder diagram to circuit design language, Int. J. Adv. Manuf.Technol. 49 (5–8) (2010) 615–626.
[45] S. Ichikawa, M. Akinaka, H. Hata, R. Ikeda, H. Yamamoto, An fpga implementation of hard-wired sequence control system based on plc software, IEEJ Trans. Electr. Electron. Eng. 6 (4) (2011) 367–375.
[46] D. Gawali, V.K. Sharma, Fpga based micro-plc design approach, Advances in Computing, Control, Telecommunication Technologies, 2009. ACT '09. International Conference on, (2009), pp. 660–663.
[47] Z. Hajduk, B. Trybus, J. Sadolewski, Architecture of fpga embedded multiprocessor programmable controller, Ind. Electron. IEEE Trans. 62 (5) (2015) 2952–2961.
[48] M.D. Hill, M.R. Marty, Amdahl's law in the multicore era, Computer 41 (7) (2008).
[49] L. Ferrarini, C. Veber, Implementation approaches for the execution model of iec 61499 applications, Industrial Informatics, 2004. INDIN '04. 2004 2nd IEEE International Conference on, (2004), pp. 612–617.
[50] G. Ochoa-Ruiz, O. Labbani-Narsis, E. Bourennane, S. Cherif, S. Meftali, J. Dekeyser, Facilitating ip deployment in a marte-based mde methodology using ip-xact: a xilinx edk case study, Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on, (2012), pp. 1–8.
[51] P.-A. Hsiung, C.-H. Huang, Y.-H. Chen, Hardware task scheduling and placement in operating systems for dynamically reconfigurable soc, J. Embedded Comput. 3 (1) (2009) 53–62.
[52] B. Navas, I. Sander, J. berg, The recoblock soc platform: a flexible array of reusable run-time-reconfigurable ip-blocks, Design, Automation Test in Europe Conference (DATE), 2013, (2013), pp. 833–838.
[53] G. Ochoa-Ruiz, S. Guillet, F. de Lamotte, E. Rutten, E.-B. Bourennane, J.-P. Diguet, G. Gogniat, An mde approach for rapid prototyping and implementation of dynamic reconfigurable systems, ACM Trans. Des. Autom. Electron. Syst. 21 (1) (2015).
[54] D. Kirschberger, H. Flatt, J. Jasperneite, An architectural approach for reconfigurable industrial i/o devices, ReConFigurable Computing and FPGAs, 2014 International Conference on, (2014), pp. 1–6.
[55] Xilinx, Partial reconfiguration overview, wp374, 2012, http://www.xilinx.com.
[56] J.P. Delahaye, J. Palicot, C. Moy, P. Leray, Partial reconfiguration of fpgas for dynamical reconfiguration of a software radio platform, 2007 16th IST Mobile and Wireless Communications Summit, (2007), pp. 1–5, https://doi.org/10.1109/ISTMWC.2007.4299250.
[57] P. Manet, D. Maufroid, L. Tosi, G. Gailliard, O. Mulertt, M. Di Ciano, J.-D. Legat, et al., An evaluation of dynamic partial reconfiguration for signal and image processing in professional electronics applications, EURASIP J. Embedded Syst. 2008 (2008) 1:1–1:11.
[58] G. Brebner, The swappable logic unit: a paradigm for virtual hardware, Field-Programmable Custom Computing Machines, 1997. Proceedings., The 5th Annual IEEE Symposium on, (1997), pp. 77–86.
[59] C. Beckhoff, D. Koch, J. Torresen, Design tools for implementing self-aware and fault-tolerant systems on fpgas, ACM Trans. Reconfigurable Technol. Syst. 7 (2) (2014) 14:1–14:23.
[60] D. Koch, C. Beckhoff, J. Teich, Recobus-builder: a novel tool and technique to build statically and dynamically reconfigurable systems for fpgas, Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on, (2008), pp. 119–124, https://doi.org/10.1109/FPL.2008.4629918.
[61] M. Touiza, G. Ochoa-Ruiz, E. Bourennane, A. Guessoum, K. Messaoudi, A novel methodology for accelerating bitstream relocation in partially reconfigurable systems, Microprocess. Microsyst. 37 (3) (2013) 358–372.
[62] G. Ochoa-Ruiz, P. Wattebled, M. Touiza, F.D. Lamotte, E.-B. Bourennane, S. Meftali, J.-L. Dekeyser, J.-P. Diguet, A modeling front-end for seamless design and generation of context-aware dynamically reconfigurable systems-on-chip, J. Parallel Distrib. Comput. 112 (2018) 1–19.
[63] M.B. Hammouda, P. Coussy, L. Lagadec, A design approach to automatically generate on-chip monitors during high-level synthesis of hardware accelerator, Great Lakes Symposium on VLSI 2014, GLSVLSI '14, Houston, TX, USA - May 21, - 23, 2014, (2014), pp. 273–278.
[64] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, B. Hutchings, Rapidsmith: do-it-yourself cad tools for xilinx fpgas, Field Programmable Logic and Applications (FPL), 2011 International Conference on, (2011), pp. 349–355.
[65] P. Garcia, K. Compton, M. Schulte, E. Blem, W. Fu, An overview of reconfigurable hardware in embedded systems, EURASIP J. Embedded Syst. (2006).
[66] K. Compton, A. DeHon, Chapter 11: Operating System Support for Reconfigurable Computing, Reconfigurable Comnputing: The Theory and Pactice of FPGA-based Computation, Morgan Kaufmann, 2007.
[67] H. Kalte, G. Lee, M. Porrmann, U. Ruckert, Replica: a bitstream manipulation filter for module relocation in partial reconfigurable systems, 19th IEEE International Parallel and Distributed Processing Symposium, (2005), https://doi.org/10.1109/IPDPS.2005.380. 151b–151b.
[68] D. Koch, J. Torresen, C. Beckhoff, D. Ziener, C. Dennl, V. Breuer, J. Teich, M. Feilen, W. Stechele, Partial reconfiguration on fpgas in practice - tools and applications, ARCS Workshops, (2012), pp. 297–319.
[69] S. Yousuf, A. Gordon-Ross, Dapr: design automation for pr fpgas. ERSA, (2010), pp. 97–103.
[70] A. Sohanghpurwala, P. Athanas, T. Frangieh, A. Wood, Openpr: an open-source partial-reconfiguration toolkit for xilinx fpgas, Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on, (2011), pp. 228–235.
[71] A. Otero, E. de la Torre, T. Riesgo, Dreams: a tool for the design of dynamically reconfigurable embedded and modular systems, 2012 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2012, Cancun, Mexico, December 5–7, 2012, (2012), pp. 1–8.
[72] N. Marques, H. Rabah, E. Dabellani, S. Weber, A novel framework for the design of adaptable reconfigurable partitions for the placement of variable-sized ip cores, Embedded Syst. Lett. IEEE 6 (3) (2014) 45–48, https://doi.org/10.1109/LES.2014.2317254.
[73] Xilinx, Vivado design suite, 2018b, https://www.xilinx.com/products/design-tools/vivado.html.
[74] R. Oomen, T. Nguyen, A. Kumar, H. Corporaal, An automated technique to generate relocatable partial bitstreams for xilinx fpgas, 2015 25th International Conference on Field Programmable Logic and Applications (FPL), (2015), pp. 1–4, https://doi.org/10.1109/FPL.2015.7293980.
[75] B. White, B. Nelson, Tincr: a custom cad tool framework for vivado, 2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14), (2014), pp. 1–6, https://doi.org/10.1109/ReConFig.2014.7032560.
[76] J. Rettkowski, K. Friesen, D. Ghringer, Repabit: automated generation of relocatable partial bitstreams for xilinx zynq fpgas, 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig), (2016), pp. 1–8, https://doi.org/10.1109/ReConFig.2016.7857186.
[77] Xilinx, Isolation design flow for xilinx 7 series fpgas or zynq-7000 ap soc (ise tools) xapp1086, 2018a, https://www.xilinx.com/support/documentation/application_notes/xapp1086-secure-single-fpga-using-7s-idf.pdf.
[78] Xilinx, Vivado design suite user guide - partial reconfiguration ug909, 2016, https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_4/ug909-vivado-partial-reconfiguration.pdf.
[79] S. Guillet, F. de Lamotte, N. Le Griguer, E. Rutten, G. Gogniat, J.-P. Diguet, Extending uml/marte to support discrete controller synthesis, application to reconfigurable systems-on-chip modeling, ACM Trans. Reconfigurable Technol. Syst.

7 (3) (2014).
[80] E. Lübbers, M. Platzner, Reconos: multithreaded programming for reconfigurable computers, ACM Trans. Embed. Comput. Syst. 9 (1) (2009) 8:1–8:33.
[81] J.-Y. Mignolet, V. Nollet, P. Coene, D. Verkest, S. Vernalde, R. Lauwereins, Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip, Proceedings of the Conference on Design, Automation and Test in Europe, DATE, (2003).
[82] X. Iturbe, K. Benkrid, C. Hong, A. Ebrahim, R. Torrego, I. Martinez, T. Arslan, J. Perez, R3tos: A novel reliable reconfigurable real-time operating system for highly adaptive, efficient, and dependable computing on fpgas, Comput. IEEE Trans. 62 (8) (2013) 1542–1556.
[83] D. Göhringer, S. Werner, M. Hübner, J. Becker, Rampsocvm: runtime support and hardware virtualization for a runtime adaptive mpsoc, International Conference on Field Programmable Logic and Applications, FPL 2011, September 5–7, Chania, Crete, Greece, (2011).
[84] A.K. Jain, K.D. Pham, J. Cui, S.A. Fahmy, D.L. Maskell, Virtualized execution and management of hardware tasks on a hybrid arm-fpga platform, J. Signal Process. Syst. 77 (1–2) (2014) 61–76, https://doi.org/10.1007/s11265-014-0884-1.
[85] G. Doukas, K. Thramboulidis, A real-time-linux-based framework for model-driven engineering in control and automation, Ind. Electron. IEEE Trans. 58 (3) (2011) 914–924.
[86] J. Dondo, F. Rincon, J. Barba, F. Moya, F. Sanchez, J. Lopez, Persistence management model for dynamically reconfigurable hardware, Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on, (2010), pp. 482–489.
[87] H. Kalte, M. Porrmann, Context saving and restoring for multitasking in reconfigurable systems, Field Programmable Logic and Applications, 2005. International Conference on, (2005), pp. 223–228.
[88] P. Wattebled, J.-P. Diguet, J. Dekeyser, Membrane-based design and management methodology for parallel dynamically reconfigurable embedded systems, ReCoSoC, (2012), pp. 1–8.
[89] H. Giesen, B. Gojman, R. Rubin, J. Kim, A. Dehon, Continuous online self-monitoring introspection circuitry for timing repair by incremental partial-reconfiguration (cosmic trip), ACM Trans. Reconfigurable Technol. Syst. 11 (1) (2018) 3:1–3:23.
[90] A. Gupte, P. Jones, Hotspot mitigation using dynamic partial reconfiguration for improved performance, Reconfigurable Computing and FPGAs, 2009. International Conference on, (2009), pp. 89–94.
[91] D. Montminy, R. Baldwin, P. Williams, B. Mullins, Using relocatable bitstreams for fault tolerance, Adaptive Hardware and Systems, 2007. Second NASA/ESA Conference on, (2007), pp. 701–708.
[92] J. Noguera, R.M. Badia, Multitasking on reconfigurable architectures: microarchitecture support and dynamic scheduling, ACM Trans. Embedded Comput. Syst. 3 (2) (2004) 385–406.
[93] K. Papadimitriou, A. Dollas, S. Hauck, Performance of partial reconfiguration in fpga systems: a survey and a cost model, ACM Trans. Reconfigurable Technol. Syst. 4 (4) (2011).
[94] J. Resano, J. Clemente, C. Gonzalez, D. Mozos, F. Catthoor, Efficiently scheduling runtime reconfigurations, ACM Trans. Des. Autom. Electron. Syst. 13 (4) (2008) 58:1–58:12.
[95] E. Hung, F. Eslami, S.J.E. Wilton, Escaping the academic sandbox: realizing vpr circuits on xilinx devices, 2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines, (2013), pp. 45–52, https://doi.org/10.1109/FCCM.2013.40.
[96] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K.B. Kent, J. Anderson, J. Rose, V. Betz, Vtr 7.0: next generation architecture and cad system for fpgas, ACM Trans. Reconfigurable Technol. Syst. 7 (2) (2014) 6:1–6:30.

Gilberto Ochoa Ruiz received his PhD degree from the LE2I Laboratory at the University of Burgundy, France, in 2013. Afterwards, he was a post-doctoral research fellow at Lab-STICC laboratory during the period 214–2015, working in the use of reconfigurable devices and architectures for cyber-physical systems. Currently, he is a an Associate Professor at Universidad Autonoma de Guadalajara, in which he works in the creation of cyber-physical systems and on embedded vision systems based on reconfigurable devices. He can be reached at gilberto.ochoa@edu.uag.mx.

Lina M. Aguilar-Lobo received her PhD degree in Electrical Engineering with the especially in Electronic Design from CINVESTAV Guadalajara, in 2016, and her M.S. degree in Electrical & Computer Engineering from the University of Guadalajara. Currently, she is researcher at the Computer Science Department of the Universidad Autónoma de Guadalajara. Her research interests are modeling and linearization of RF power amplifiers, wireless communication systems, real-time embedded systems, machine learning, intelligent systems and the applications of the Internet of Things. She can be reached at lina.aguilar@edu.uag.mx.

Romain Bévan received his PhD degree from the Lab-STICC laboratory at the University of South Brittany in 2013, working on model-based design tools for industrial reconfigurable systems based in the IEC 61131-3 standard (collaboration with Syleps company). Afterwards, he became adjunct lecturer at the ENSIBS Engineering School of the UBS, working in the mechatronics engineering and automation. He is also associate research engineer at Lab-STICC and at the UBS Composite Research Center ComposiTIC, where he carries out both academic and applied research in projects pertaining 3D printing, robotics, automation, embedded systems and real-time control systems. He can be reached at romain.bevan@univ-ubs.fr.

Florent de Lamotte received his PhD degree from the Lab-STICC laboratory at the University of South Brittany in 2007, working on model-based design tools for industrial reconfigurable systems. Afterwards, he became Associate Professor at the ENSIBS Engineering School of the UBS, working in the mechatronics engineering and automation. He is also associate researcher at Lab-STICC, in which he works in a variety of research interests, including model-based design tools (both for industrial applications and for modelling partially reconfigurable systems on FPGA), real-time control systems, cyber-physical systems, and electronic system design. He can be reached at florent.lamotte@univ-ubs.fr.

Jean-Philippe Diguet is a CNRS director of research at Lab-STICC, Lorient / Brest, France. He received the Ph.D. degree from Rennes University (France) in 1996. In 1997, he has been a visitor researcher at IMEC (Belgium). He has been an associate professor at UBS University (France) until 2002. In 2003, he co-funded the dixip company in the area of wireless embedded systems. Since 2004 he is a CNRS researcher at Lab-STICC, where he is heading the MOCS team. He has been a visitor researcher at the University of Queensland, Australia in 2010, an invited Prof. at Tohoku University, Japan in Nov. 2014 and at Univ. of São-Paulo, Brazil, in Nov. 2016. His current work focuses on various aspects of embedded system design: Designs and Tools for NoC-based MPSoC architectures, self-adaptivity for uncertain environments and customized hardware architectures including security and embedded vision. Jean-philippe.-diguet@univ-ubs.fr.