Embedded Runtime Reconfigurable Nodes for Wireless Sensor Networks Applications

Yana Esteves Krasteva, Jorge Portilla, Eduardo de la Torre, and Teresa Riesgo

***Abstract*—The use of reconfigurable hardware (HW) can improve the processing performance of many systems, including Wireless Sensor Networks (WSNs). Moreover, reconfigurable devices permit remote and runtime HW reconfiguration, which implies benefits in WSNs deployment and maintainability and, finally, cost reduction.**

In this paper, WSN node runtime reconfigurability is tackled from several aspects. First, the sensor node includes a commercial reconfigurable device, a Field Programmable Gate Array (FPGA), that permits to take advantage of the tools and support provided by the industry, while exploiting the inherent hardware parallelism. Second, two software (SW) and hardware reconfiguration scenarios are defined along with a support middleware. Third, in order to provide runtime reconfigurability to the WSN node, a complete runtime reconfigurable system has been defined and designed for the FPGA included in the node. Fourth, the HW reconfiguration cost has been evaluated, as well as the cost of transmitting new HW configurations and SW programs through the network, based on a set of defined parameters. Finally, the feasibility of the runtime reconfigurable system has been demonstrated with a use case.

***Index Terms*—Embedded systems, field programmable gate arrays (FPGAs), reconfigurable architectures, wireless sensor networks (WSNs).**

I. Introduction

WIRELESS SENSOR NETWORKS (WSNs) represent one of the most challenging areas in today's electronic industry [1], [2]. These networks are expected to be autonomous, low-power demanding, context aware, and flexible. A final application may have hundreds or thousands of sensor nodes spread out in an environment, making the deployment and the support of WSNs a complex task. Although the integration technologies are clearly tending to smart sensor, there is still an increasing variety of sensors. The interfaces and data processing required for sensors control are very different not only from one sensor to another, but also from one application to another. In such context, the use of identical nodes or a reduced set of nodes that are further adapted and/or customized would simplify the deployment process and reduce the product cost.

Manuscript received September 06, 2010; revised December 03, 2010; accepted December 31, 2010. Date of publication January 10, 2011; date of current version July 27, 2011. The associate editor coordinating the review of this paper and approving it for publication was Prof. Ralph Etienne-Cummings.

The authors are with the Centro de Electrónica Industrial, Universidad Politécnica de Madrid, Jose Gutierrez Abascal 2, 28006, Madrid, Spain (e-mail: yana.ekrasteva@upm.es; jorge.portilla@upm.es; eduardo.delatorre@upm.es; teresa.riesgo@upm.es).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSEN.2011.2104948

Classic design approaches for WSNs node rely on the use of a microcontroller (μC) [3], [4]. However, processing and functionality needs are continuously increasing due to new application requirements and this includes WSNs applications. This, added to the constantly increasing pressure for reducing time-to-market, has led to new design alternatives, such as reconfigurable hardware (HW).

Reconfigurable systems have been studied in the last years as an alternative for both: Application-Specific Integrated Circuits (ASICs) and General Purpose Processors (GPPs). GPPs, for instance, suffer an imbalance between IO (device Input-Outputs) and processing. Differently, FPGAs provide not only a large amount of IOs, but also high processing due to the offered inherited of parallelism and pipelining. Furthermore, reconfigurable systems provide high flexibility as they can be updated after system deployment and also permit time to market reduction, since the proper prototype can be the final device. As a result, reconfigurable devices are constantly gaining market share and extending their industry application domains and research interest [5], [6]. For instance, the work presented in [7] focuses on the application of FPGAs in industry control systems. Even more, now, it is possible to implement neural networks and fuzzy control solutions on FPGAs [8] and also, reconfigurable computing has been widely explored in accelerating applications, like calculations related to molecular dynamics in [9].

This paper is focused on the use of reconfigurable systems in WSNs. Several research groups have already exploited the benefits of HW parallelism by designing ad-hoc reconfigurable devices prepared to be adapted to a set of prerecorded applications, like in [10] and [11]. The flexibility achieved with this approach is higher compared to ASIC-based solutions, but not as high as with small grain reconfigurable devices, like FPGAs. In addition to this, custom solutions are more restricted as they require specific Computer Aided Design (CAD) tools. Differently, this paper exploits industry available reconfigurable devices looking for high adaptability and flexibility through the use of the partial runtime reconfiguration technique, while taking advantage of the vendor provided CAD tools. In particular, this paper aims to explore and evaluate the use of partial runtime reconfiguration in WSNs, topic that has not been explored in the state-of-the-art.

Runtime reconfiguration is an advanced topic within the reconfigurable computing area, where changes into the FPGA configuration are done at runtime, while the device I/Os and remaining logic is kept active. This powerful feature (only included in Xilinx and Atmel FPGAs) permits not only to perform HW updates at runtime and at anytime, but also to save memory space and programming time compared to full FPGA reconfiguration. Partial reconfiguration has already been

exploited in the automotive industry [12] and in shape-adaptive video applications [13], where different reconfiguration strategies are studied.

Runtime reconfiguration will be tested in a HW platform for WSNs developed at CEI, called Cookies and first presented in [14]. This platform is overviewed in the paper, along with an outline of the WSNs state-of-the-art, in order to highlight its distinguishing characteristics.

This paper describes the entire reconfigurable system design that includes: the definition of the node reconfigurability scenarios, the support SW definition and implementation and, a detailed description of the partial runtime reconfigurable system definition and design process. In order to test the designed system, a work flow, also described in this paper, has been used to generate node HW configurations. Furthermore, a set of general parameters have been defined to evaluate the system reconfiguration cost and applied to the target node. Finally, a partial runtime reconfiguration use case has been created to validate the feasibility of remote runtime reconfiguration in WSNs.

The rest of this paper is organized as follows. Section II includes the review of the available industrial and academic WSN along with the brief description of the Cookie modular node architecture. The identified node reconfiguration scenarios, along with a general view of a reconfiguration control middleware, are included in Section III. Section IV presents the design of the proposed partial runtime reconfiguration system for WSN and the complete reconfiguration workflow. A system evaluation based on general reconfiguration cost parameters and the results can be found in Section V. A discussion of some reconfigurable system aspects related to WSN nodes can be found in Section VI. Finally, conclusions are included in Section VII.

II. Cookie Sensor Node

In this section, a general state-of-the-art in HW platforms for WSNs nodes is presented. Both industrial and academic approaches are presented. The Cookie node is described in Section II-B.

A. WSNS Hardware Platforms State-of-the-Art

Several HW platforms with different approaches have been developed by different research groups during the last years. Most of them present a SW point-of-view in which the core of the processing stage is a microcontroller. Researchers from UC Berkeley developed the TelosB sensor node, one of the widely used platforms for WSN research and development [3]. This platform is based on an ultra-low-power microcontroller and a low data rate low-power IEEE 802.15.4 [15] compliant radio chip. Sensors can be added to the platform through an expansion connector. Hitachi has developed a sensor node with a modular approach [4], based on a microcontroller resized for dimension reduction, proprietary battery technology and IEEE 802.15.4 compliant radio chip.

These are two examples of SW-based platforms for WSNs. Although they are very good solutions, they are resource limited, so in high data processing applications, problems related to overhead can occur.

Traditionally, reconfigurable devices like Complex Programmable Logic Devices (CPLDs) or FPGAs are not included in designs for sensor nodes, mainly due to their high power consumption. On the other hand, solutions in which such devices are integrated can target a wider set of computationally intensive applications, as well as adding flexibility to the sensor node. For example, in the platform used in the present work, a Spartan 3 FPGA from Xilinx and an ADuC841 microcontroller from Analog Devices are used to carry out the processing tasks. There is a processing layer version, to be available in the near future, composed of a MSP430 microcontroller from Texas Instruments and an Actel Igloo FPGA. This low-power version will consume 10 μA in sleep mode, which is a real competitive power consumption value compared with the state-of-the-art platforms. As a reference, the Cookie node version used in this paper consumes 30 mA, while the widely used Crossbow TelosB platform that does not include HW reconfigurability consumes 5.1 μA (in standby mode).

However, some solutions for WSN nodes include reconfigurable HW elements. A modular reconfigurable platform has been developed by Microsoft Research Labs [16]. In this platform, several layers can be added to the final platform, with several microprocessors depending on the application requirements, and the reconfigurable HW (CPLD in this case) is used to achieve communication abstraction through the entire modular platform. Therefore, the processors within the platform communicate directly with the CPLD, which makes possible different final implementations. Tyndall National Institute has developed another modular platform [17], which allows the inclusion of an FPGA layer to carry out Digital Signal Processing (DSP) tasks. They make a division between processing, sensing, communication, and power supply, which is the way in which the sensor nodes functionality is divided.

The groups described in the previous paragraph use commercial reconfigurable devices in their designs. On the other hand, other groups work in developing new integrated circuits, which incorporate reconfigurable HW as part of the entire circuit. In [10], a specific integrated circuit with a microcontroller unit and a reconfigurable part is presented. Their goal is to provide the node with flexibility in order to adapt it to environment changes. In [11], an approach in which the system adapts itself to execute the tasks more efficiently, in terms of power, is presented.

B. Cookie Sensor Node

The distinguishing characteristic in Cookie is that it has been designed in order to provide flexibility and adaptability, by applying modularity at the physical HW level through a layered printed circuit board (PCB) structure (see Fig. 1) and providing it with reconfigurable HW (FPGA).

Modularity allows dividing and encapsulating the functionality included in the node. Therefore, future redesigns may involve only part of the platform, which is desirable considering time-to-market and evolving technologies and standards. Moreover, due to the node flexibility, it is possible to carry out a design space exploration for the HW node, by interchanging different implementations for each layer, depending on the application.

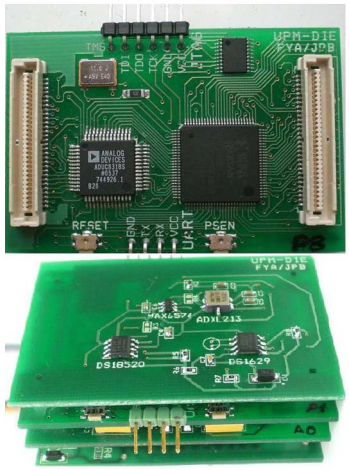

Fig. 1. On the top part, the Cookie processing layer (uC on the left and the FPGA on the right) and on the bottom, a complete Cookie node.

The Cookie is composed of four main layers (more layers can be added in future versions): processing, communications, power supply, and sensors. Every layer carries out a specific task. In the following paragraphs, these layers are detailed.

1) Processing: This is the heart of the node. It includes an 8051 microcontroller core enhanced with several peripherals from Analog Devices (ADuC841) and a Xilinx XC3S200 Spartan 3 FPGA. The microcontroller and the FPGA share three 8-bit ports for communication. Part of one port of the microcontroller is connected to the Join Test Action Group (JTAG, IEEE 1149.1 [18] standard) programming port of the FPGA.
2) Communications: The last version of this layer includes a ZigBee module (ETRX2 from Telegesis). This module is managed by the microcontroller through the Universal Asynchronous Receiver Transmitter (UART) port. Another layer version with Bluetooth is also available.
3) Power supply: This layer generates all voltages needed within the Cookie. Two versions have been developed. The one used in the work presented in this paper includes a Universal Serial Bus (USB) connection, which allows power supply from a PC, as well as serial programming for the μC.
4) Sensors: This layer includes those elements which are intended to take measurements from the environment. Three different layers have been developed for the Cookie. These layers include sensors of acceleration, temperature, humidity, light, infrared, and deformation (strain gauge).

Additionally, other layers can be added to the modular platform like, for instance, a memory layer suitable for expanding the microcontroller program memory and/or for holding a HW configuration library.

One of the features of the sensor layer is that it can include sensors with both digital and analog interfaces (they will be called digital sensors and analog sensors respectively from now on). In Fig. 2, the architecture of the system is shown. Signals from analog sensors are connected to the ADC of the microcontroller. On the other hand, signals from digital sensors are connected to the FPGA. In principle, the FPGA carries out all

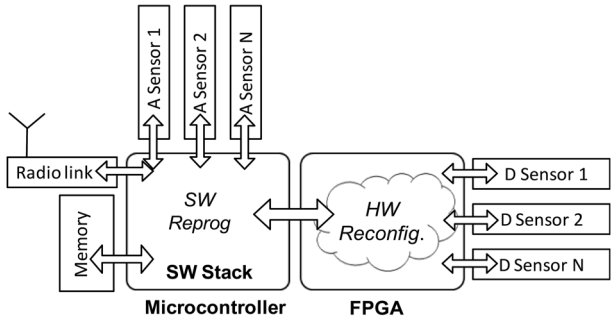

Fig. 2. Sensor node system architecture. Analog sensors are connected to the uC ADCs, while digital sensors are connected to the FPGA.

processing related to digital sensors in order to release the μC that is in charge of managing the node communications and processing data from analog sensors. Nevertheless, if an external ADC is included in a sensor layer version, analog sensors could be directly connected to the FPGA.

Nowadays, there are a myriad of sensors in the market with several different interfaces. Many of them are digital sensors, with different protocols such as SPI, I2C, 1-Wire, etc. When this kind of signals have to be processed using a microcontroller, problems related to timing and processor overhead can appear. In fact, some manufacturers offer Hardware Description Language (HDL) code to implement the sensor interfaces in a coprocessor, for instance, Dallas Semiconductor with its 1-Wire interface [19].

In this context, a proprietary library of generic HW interfaces [20] has been developed in order to process sensor signals with very different digital interfaces. Currently, the interfaces included in the library are as follows.

- I2C and I2C modified (interface for Sensirion Company sensors).
- PWM.
- Period/Frequency.
- 1-Wire.

The library is divided in modules which represent sensor interfaces (actuator interfaces can be added as well). Every module has been designed following a philosophy inspired in the IEEE 1451 family of standards, but they can also be used without being compatible to them. Each sensor or actuator (transducer) is "seen" as a channel (or set of channels) by the transducer controller. Two channel types are recognized: sensor channel and actuator channel. Some sensors, like the SHT11 from Sensirion, supply two or more measures (in this case, humidity and temperature). Therefore, for the same sensor, two different channels are needed.

The microcontroller sends triggers to the FPGA, specifying the sensor from which the measure has to be taken, and the FPGA activates the appropriate sensor interface. Then, the FPGA sends the result to the microcontroller. The FPGA acts as a reconfigurable coprocessor for the microcontroller by taking the signals from the digital sensors, and processing the information for the microcontroller. In other cases, this coprocessing cannot be made by the microcontroller, such as the case of relatively complex filtering stages, which could not be covered due to data bandwidth restrictions.

These interfaces along with HW coprocessing in the FPGA will be used to show the runtime reconfiguration capabilities of the platform, using partial reconfiguration techniques.

III. Node Reconfiguration Scenarios and Control SW

In this paper, node reconfigurability is associated not only with loading new SW programs in the microcontroller (SW reprogramming), but also new HW configurations in the reconfigurable fabric (HW reconfiguration). Related to this, two general reconfiguration scenarios have been differentiated and applied to the Cookie node.

- The first one covers *reconfiguration at network level* that is mostly required during deployment, where the final function of each network node is defined (this includes the used sensors and data processing) or, when the network function is changed (like in an emergency situation). Generally this scenario may cover both, SW reprogramming and HW reconfiguration. In the context of the Cookie platform, HW reconfiguration is mandatory when modifications affect digital sensors (marked with "D" in Fig. 2), as they are connected directly to the FPGA. Contrary, SW reprogramming is required when analog sensors (marked with "A" in Fig. 2) are involved, because they are directly connected to the microcontroller ADCs. Furthermore, Cookie runtime reconfiguration in this scenario is possible only when dealing with HW reconfiguration, because SW reprogramming requires rebooting the system.
- The second scenario, *reconfiguration at node level*, mainly involves HW reconfiguration. In this case, a reconfigurable array acts as a reconfigurable coprocessor where computation intensive tasks take advantage of the HW parallelism in order to lighten the microcontroller, like in [10]. With respect to the Cookie platform, when dealing with analog sensors, the entire FPGA is perceived as a coprocessor. Different, when dealing with digital sensors, two functionalities have to be allocated in the FPGA (the coprocessor and the digital sensors control).

In order to cover both scenarios, a partial runtime reconfiguration system has been built on top of the Cookie node FPGA and partial reconfiguration has been used as a reconfiguration technique. The resulting runtime reconfigurable system permits to independently modify (while the remaining system is running): the digital sensor interfaces, the data processing and/or the FPGA to microcontroller interface. The partial reconfigurable system design and implementation is the main topic of Section IV (next, the focus is taken back on the overall node reconfigurability).

The microcontroller is the core element in the node reconfigurability control. It is in charge of receiving new HW configurations and SW programs, manage them and deal with the FPGA reconfiguration. Rather than extending an operating system to support reconfigurability, like in [21] and [22], the goal within this work has been to keep the system as simple as possible. Therefore, the node reconfigurability is controlled by a simple and independent software stack, briefly defined next and shown in Fig. 3.

- On top of the node hardware, shown in Fig. 3 with dashed lines, the first layer is an abstraction layer. Abstraction layers are defined to facilitate applications porting across different platforms. In a reconfigurable system, the abstraction layer is composed of two parts: one is the traditional Hardware Abstraction Layer (HAL) and the second one is a Reconfigurable Hardware Abstraction Layer (RHAL). For the Cookie, the RHAL includes: i) a definition of the partial runtime reconfigurable system architecture along with FPGA device descriptors (FPGA provider, available logic, etc.) that are stored in a set of bits; ii) a HW configuration library, that includes a set of partial configuration files that can be loaded into the FPGA; and iii) the FPGA programmer, the software that is in charge of programming the FPGA.
- Node descriptors and state variables: the descriptors define the node currently available resources, like radio communication standards, sensors, and computational resources located in libraries (HW and/or SW), while state variables define the node resources that are currently used.
- The control layer includes the reconfiguration policy that is directly related to the node decision taking independency. Decisions are basically related to two main aspects: when a node reconfiguration is required, or what to do if the required configuration is not available.
- The application layer defines the node current goal (sensing, retransmitting, data processing, etc.).

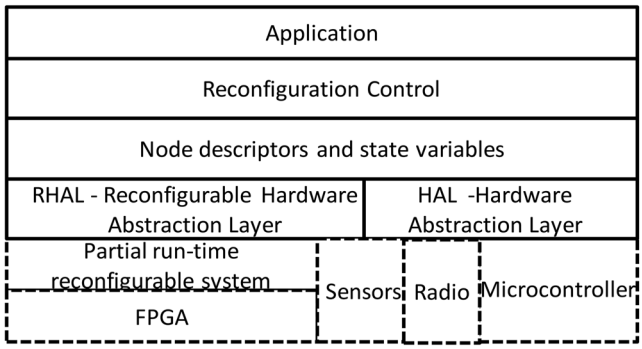

Fig. 3. Reconfigurable node software stack that runs on the node microcontroller on top of the node hardware (dashed lines).

Up to now, following the software stack previously described, a simple stack implementation that runs on the node μC has been set up in order to evaluate the delivery of new node configurations using the network. Some relevant implementation details are described in the following paragraphs.

Regarding the FPGA programmer, the available Spartan3 FPGA does not have an Internal Configuration Access Port (ICAP) which is a better option for partial runtime reconfigurable systems given that, with it, the system is able to control its own reconfiguration process. From the remaining configuration options, which are Select Map and JTAG. JTAG has been selected because it is supported by most of the commercially available FPGA devices. A SW implementation of a JTAG controller, provided by Xilinx, runs on the microcontroller and acts as the FPGA programmer. As a result, no FPGA area is required for controlling the reconfiguration process, opposite to [23], where a Select Map controller is implemented in HW (with FPGA logic). On the other hand, the main disadvantage

of using JTAG is the reconfiguration time (JTAG is serial, while Select Map is parallel).

Regarding the node descriptors and state variable, a unique code has been assigned to each HW configuration that allows the identification of what processing, interface and/or sensing configurations are currently loaded in the FPGA. This code can be read and retransmitted trough the network by the microcontroller.

On top of this, the defined node control is quite simple, and it is in charge of receiving and/or sending new SW programs and HW configurations through the network, and loading them in the μC program memory.

Regarding network data transmissions, a ZigBee communication module has been used and several tests have been done. Evaluation results can be found in Section V.

IV. Partial Runtime Reconfigurable System for WSN

The main advantage of exploiting partial reconfiguration in embedded devices, like WSN nodes, apart from the possibility of changing part of the configuration in runtime, is that partial configuration files are much smaller than complete ones, and this produces lower memory and bandwidth requirements (this will be demonstrated in the results Section V).

In order to design a partial runtime reconfigurable system, the FPGA array is divided into reconfigurable and fixed regions. The resulting architectural division is usually called FPGA structure, resource arrangement or partitioning [24]. In this paper, the FPGA partial reconfigurable system has been built based on *Virtual Architectures (VAs)* that integrate both the resource division and the on-chip communications (the on-chip interconnection of the different FPGA regions). This section goes briefly through the main general steps for designing VAs, presented in [25], particularized to the target Spartan 3 FPGA (the one available in the Cookie sensor node) and, after that, the workflow used for the reconfiguration process is described.

The first aspect to be taken into account for VAs is the model and the reconfiguration granularity that are going to be supported. VA models can be one dimensional (1D) or two dimensional (2D). Partial reconfiguration in Virtex II-based architectures (including Spartan 3 FPGAs) is frame-based [26], where entire columns of logic elements are reconfigured. Thus, 1D-based VAs are well suited. In order to facilitate the reconfiguration process of the system, the reconfiguration granularity selected is coarse grain, since a reconfiguration implies a full IP Core (Intellectual Property Core) to be loaded in the FPGA. In the context of reconfigurable systems, these cores represent already placed and routed designs (partial configuration files) and are referred as *hard cores* [27].

The next step is to analyze the FPGA structure in order to cover the requirements of granularity and architecture model. The structure of the Spartan 3 FPGA is quite regular compared to other FPGAs (Virtex II for instance). Therefore, the fixed area of the FPGA occupies the leftmost and rightmost sides of the FPGA array, spanning the area enclosed between the left/right IOs and the embedded memory columns, marked with black rectangles in Fig. 4. The fixed area on the left side is used to access the external microcontroller, while the right side fixed area is used to access the node sensors (see Fig. 4). The middle FPGA

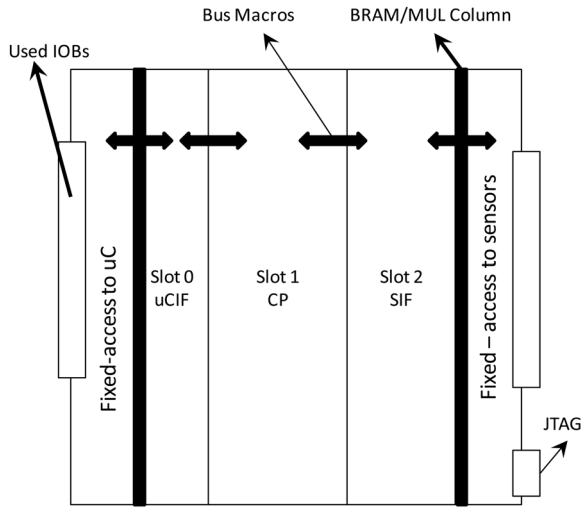

Fig. 4. Sensor node Spartan 3 FPGA Virtual Architecture (VA)—schematic view. The VA is composed of three slots with different width and a pipeline like on-chip communication build by three types of bus macros.

section that is enclosed between the two memory columns has a very regular structure, and is defined as the reconfigurable area. This area is divided into *Slots*. A slot is defined as a group of FPGA Configuration Logic Block (CLB) columns. CLBs are the main reconfigurable components. Each CLB is composed of four LookUp Tables (LUTs).

In the target Spartan 3 FPGA (XC3S200), the resulting reconfigurable area spans throughout 16 CLB columns, with 24 CLB rows. A pipeline-like triple slot distribution, where each slot has a special function, has been selected. Runtime reconfiguration support applies [28] to three different functions: sensors interface, reconfigurable coprocessor and μ C-to-FPGA communication, and three different slot widths have been defined for them, which are 2, 8, and 6 slots, respectively, according to generic core size estimations. Slot 0 defines the communication with the microcontroller (microcontroller interface, referenced as uCIF in Fig. 4), slot 1 is used to allocate the reconfigurable coprocessor (CP in the figure) and finally, in slot 2, the sensor interface control (SIF) is allocated.

An important aspect in virtual architectures is the on-chip communication. In the presented approach, each slot and/or fixed region is connected to the neighboring areas to form the pipeline. This connection is done using special communication structures called Bus Macros. All signals that have to cross the slot boundaries require a Bus Macro [26]. Apart from the unidirectional Bus Macros, provided by Xilinx, two other bus macro structures have been designed. First, a bidirectional macro that passes four data bits from left to right and four bits from right to left. The second type is also a bidirectional macro, but it is used to cross the mentioned Block RAM/Multipliers columns (BRAM/MUL column in Fig. 4). In some cases, a hard core needs access to such embedded resources. In the presented VA, this can be done by hard cores loaded in either slot 2 or slot 0.

Once all the previous VA design considerations have been set up, the FPGA VA is defined by a set of files that result from the design process: i) a user constraint file (.ucf file) that mainly defines slots positions and boundaries and ii) the communication

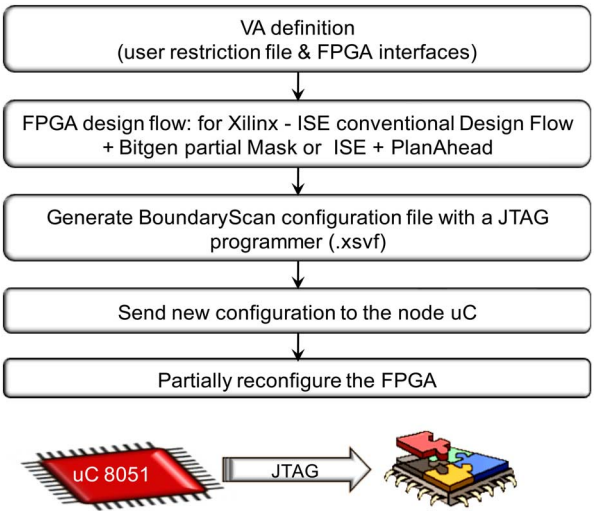

Fig. 5. Runtime WSN node partial reconfiguration working flow. The flow is used to generate and send new HW configuration to the FPGA included in the "mote" node.

macros used to build the on-chip communications, that are provided as relatively placed macros (.nmc files).

Once the VA to be used has been selected (several VAs can be defined for a single FPGA) and the related definition files identified, hard cores for the target system can be designed. This process can be done with a variety of design flows, depending on the FPGA provider. For Xilinx, one option is to use the conventional ISE (Integrated System Environment) design flow, provided by Xilinx. This flow ends with the generation of a full design netlist, from which hard cores are generated using bitgen (the Xilinx tool for bitstream generation) by applying specific partial mask options. The second approach is to use the Xilinx partial reconfiguration design flow that is based on ISE and the PlanAhead tool (suitable for testing different floorplanning approaches). Differently from the first approach, the second one directly produces partial configuration files and has better routing in most cases.

For the hard core design of the architecture presented in this paper, the first approach was selected as it permits the use of free of charge version of the tool (ISE web pack) and because the floorplanning in our design method is already included in the VA definition.

The list of the currently available cores are the following.

- Coprocessors (CP): An eight bits moving average filter that constantly collects measured data and gives an average value, and a FIR filter.
- Microcontroller Interface (uCIF): There is one version for accessing digital sensors data (uCDIF).
- Sensor interface (SIF): There are hard cores available for the temperature and the accelerometer sensors in two versions: i) one using the embedded multipliers (MULs) available in the BRAM/MUL column that correspond to slot 2, and ii) a second version that implements the multiplications in LUTs.

The complete workflow used to test partial reconfiguration in WSNs is presented in Fig. 5. As it can be noticed, the flow starts with a virtual architecture selection and then continues with the already described hard core design. After that, independently from the method used to generate partial configuration files, the next step is to use a JTAG FPGA programming tool (iMPACT for Xilinx FPGAs) to generate the required binary formatted boundary scan configuration file. After that, the next step of the workflow is to send new configurations through the network to a target node, where they are saved in a local memory and partial reconfiguration is performed.

If the target FPGA is not a Xilinx one, then, a tool called svf2xsvf [29] has to be used to generate complete configuration files, but partial reconfiguration is not possible.

V. Reconfigurable System Evaluation: Parameters Definition, Tests, and Results

Several parameters have been defined to evaluate: i) the use of reconfigurable systems in WSNs and ii) the delivery of new HW configurations and SW programs through the WSNs network. The evaluation parameters, related to the main embedded device restrictions, memory and energy, are based on relative values of cost. Cost is a way of measuring how much resources (energy and memory) are needed to perform an action. Additionally, the use of remaining energy or memory space allows having an independent evaluation system. For instance, the evaluation system could be applied to other processing layer solutions, like the Cookie power optimized version (mentioned in Section II).

The first aspect, related to energy, follows the definition of power, as the energy consumed during the time needed for finishing a certain task. Distinctions have been made below.

- Transmission Energy: The energy used during HW configurations and SW programs data transmission

$$E_{\text{trans}} = \mathrm{T}_{\text{trans}} * \mathrm{P}_{\text{trmode}} \tag{1}$$

  where $\mathrm{T}_{\text{trans}}$ is the time needed to transmit a configuration and $\mathrm{P}_{\text{trmode}}$ is the power consumption in this working mode.

The remaining definitions follow the same logic.

- Reception Energy

$$\mathrm{E}_{\text{rec}} = \mathrm{T}_{\text{rec}} * \mathrm{P}_{\text{recmode}}. \tag{2}$$

- Retransmission Energy

$$\mathrm{E}_{\text{ret}} = \mathrm{T}_{\text{ret}} * \mathrm{P}_{\text{retmode}}. \tag{3}$$

- Reconfiguration Energy is the energy needed for a single reconfiguration

$$\mathrm{E}_{\text{reconf}} = \mathrm{T}_{\text{reconf}} * \mathrm{P}_{\text{reconfmode}}. \tag{4}$$

- Standby Energy is the energy spent in standby mode

$$\mathrm{E}_{\text{sbmode}} = \mathrm{T}_{\text{sbmode}} * \mathrm{P}_{\text{sbmode}}. \tag{5}$$

  The cost of doing a task is measured as the part of the total currently available node energy (E_{node}) that has to be consumed to finish that task.

- Transmission Cost

$$C_{\text{trans}} = \frac{E_{\text{trans}}}{E_{\text{node}}}. \quad (6)$$

- Reception Cost

$$C_{\text{rec}} = \frac{E_{\text{rec}}}{E_{\text{node}}}. \quad (7)$$

- Retransmission Cost

$$C_{\text{ret}} = \frac{E_{\text{ret}}}{E_{\text{node}}}. \quad (8)$$

- Reconfiguration Cost

$$C_{\text{reconf}} = \frac{E_{\text{reconf}}}{E_{\text{node}}}. \quad (9)$$

- Total Reconfiguration Cost $C_{\text{totalreconf}}$ is the sum of the costs of all needed reconfigurations $(\text{N}_{\text{reconf}})$ when passing from one application to another.

$$C_{\text{totalreconf}} = \sum_{1}^{N_{\text{reconf}}} C_{\text{reconf}}. \quad (10)$$

Additionally, the total Remote Reconfiguration Cost (C_{TRR}) is defined as the sum of all the C_{trans}, C_{ret} and C_{rec} needed to reach the *Node Under Reconfiguration (NUR)*

$$\text{C}_{\text{trr}} = \text{C}_{\text{trans0}} + \sum_{1}^{\text{Nhop}} \text{C}_{\text{ret}} + \text{C}_{\text{recNhop+1}} \quad (11)$$

Nhop is the number of hops.

The second aspect, related to the node memory resources, is the cost of having a configuration (C_{lib}) in the local library. C_{lib} is defined as the memory space portion used by the configuration (CN_{mem}) of the total available memory in the node (N_{mem}), minus a coefficient related to the importance of this configuration CN_{imp}

$$C_{\text{lib}} = (N_{\text{mem}}/CN_{\text{mem}}) - CN_{\text{imp}}, \quad 0 \leq CN_{\text{imp}} \leq 1. \quad (12)$$

Generally, a certain task is considered as worthy to be done by a node only when the task cost, minus a coefficient related to the importance of the task, is less that a defined cost threshold (C_{trsh}). Importance coefficients related to both, memory and energy, are derived locally, on each node, depending on the WSNs main objective that is defined by a centralized administrator in charge of sorting the network priorities. Additionally, the non total costs values are between 0 and 1. Thus, a task cannot be done by the node if its cost is higher than 1.

Some of the parameters have been calculated for the Cookie node. Before applying the defined equations, the power consumption in each working mode (standby, transmitting, receiving, retransmitting, and reconfiguring) has to be measured. This, static measure, can be done with a conventional ampermeter during the sensor node consumption characterization. For instance, the Cookie node runs at 3 V and the power consumed in standby mode $(\text{P}_{\text{sbmode}})$ is 60 mW, 150 mW in reconfiguration $(\text{P}_{\text{reconfmode}})$ and 240 mW in transmission $(\text{P}_{\text{trmode}})$ and reception modes $(\text{P}_{\text{recmode}})$ and 110 mW in retransmission mode $(\text{P}_{\text{retmode}})$.

TABLE I
DATA TRANSMISSION TIME AND RATE

Trans. Mode	SW (sec.)	HW (sec.)	Total (sec.)	data rate (Bytes/s)
Cable 8B	19,06	73,5	87,22	333,04
ZigBee 8B	49,11	190,22	219,31	131,95
Cable 16B	13,72	53,56	67,28	470,70
ZigBee 16B	29,09	114,75	143,75	220,30

TABLE II
TRANSMISSION COST

Trans. Mode	E_{trans} (mWmim)	C_{trans}	E_{ret} (mWmim)	C_{ret}
SW ZigBee 8B	197	0,055	408	0,113
HW ZigBee 8B	760	0,211	1520	0,422
SW ZigBee 16B	116	0,032	232	0,064
HW ZigBee 16B	459	0,127	918	0,25

The second value that has to be measured is the time needed to perform a task (receive, retransmit, reconfigure) and can also be measured statically. Time values depend on several aspects and the most relevant (transmission media and packet sizes) are described next in the context of the Cookies. The selected time characterization testing updates correspond to: a slot 2 (six CLB columns) HW configuration and the embedded SW program used for the FPGA programming. These updates have been transmitted to a NUR using different types of transmission media: i) through the available wireless ZigBee communication and ii) using a serial cable connection. Table I summarizes both transmission times and data rates. Additionally, in the table, two different packaging formats can be differentiated: one of 16 bytes and another of 8 bytes. Both transmission packet formats have been tested in two configurations: i) a direct connection with the NUR and ii) in a long distance connection, using a node for intermediate routing (multihop). Results included in Table I have been measured and calculated under a direct connection mode (single hop).

With the measured power and time values, transmission and retransmission energies have been calculated with the previously defined equations and can be found in Table II.

Now, in order to calculate cost parameters, the node available energy has to be known. This value can be calculated dynamically by constantly subtracting the node consumption from the initially available energy. There is a solution for the node current dynamic measurement where the Cookie is in charge of calculating the consumption. Although this solution has been tested, it has not been integrated so far in the reconfigurable system.

The Cookie is powered by AA batteries of 1 Ah each that are supposed to last for at least 100 h (this time is based on the normal, nonreconfiguration, function of the node), then the node has a limit consumption of 0,02 Ah. Now, let us suppose that the node has just started working, then the currently available node energy is $\text{E}_{\text{node}} = 60\ \text{mWh} = 3600\ \text{mWmin}$.

TABLE III
TRANSMISSION COST

HW Configuration	.bit (KB)	.xsvf (KB)	C_{lib}
Slot 0 (2 CLB columns)	8,7	8,9	0,13
Slot 1 (8 CLB columns)	32,9	33,3	0,52
Slot 2 (6 CLB columns)	24,9	25,2	0,393
one Slot 2 + one BRAM/MUL column	52,3	54,5	0,851

TABLE IV
TRANSMISSION COST

SW Programs	.hex (KB)	C_{lib}
SW_ACCIF	4,4	0,06
SW_FPGA_Prog	6,4	0.1

With this data, transmission and retransmission costs have been calculated and can be found in Table II for both packet transmission formats.

Reliability in WSN applications is an important aspect, even more if the network is also used for transmitting device updates. Tests have shown that the 16 bytes transmissions fail with a high rate in long distance connections compared with 8 bytes-based transmissions that are much more reliable, but they are more costly and reach values of 0.4 for a HW configuration retransmission, that is, almost half of the available energy in the node has to be spent (see Table II). Anyway, it is well known that the ZigBee standard is intended for transmitting small amount of data, and also, the ZigBee module currently used does not permit low level protocol optimization.

With respect to the system memory cost, the FPGA partial configuration file sizes in both .bit and .xsvf format (the one transmitted in the network) are presented in Table III for hard cores that use single slots and hard cores that use a slot and a BRAM/MUL column.

Regarding SW programs, an accelerometer control interface (SW_ACCIF) that includes some data computation in the microcontroller and the FPGA programmer software (SW_FPGA_Prog) have been used as an example. Characteristics for the SW program files can be found in Table IV.

The memory currently used for keeping HW configurations and SW programs is the microcontroller program memory.

C_{lib} has been calculated for HW configurations and SW programs, taking into account that this memory is 62 KB. For simplicity, CN_{imp} has been assumed to be the lowest possible value $(CN_{imp} = 0)$.

As a reference, it is important to mention that a complete XC3S200 configuration file is 131 KB and this has an unaffordable memory cost (>1) and therefore, makes partial reconfiguration well fitted for embedded, memory restricted, devices. Also, it can be noticed that the memory cost of a HW configuration file is higher than a SW program.

Reconfiguration costs can be found in Table V. It can be noticed that the cost of transmitting configurations data is much higher than the reconfiguration cost. The table includes the reconfiguration cost for the entire FPGA. This value could be used as a reconfiguration cost threshold (C_{trsh}).

TABLE V
RECONFIGURATION COST

HW Configuration	Reconf. Time(sec)	E_{reconf}	C_{reconf}
Slot 0 (2 CLBs)	3,6	9	0,0025
Slot 1 (8 CLBs)	7,8	19,5	0,0054
Slot 2 (6 CLBs)	4,3	10,75	0,0029
Full FPGA(estimated)	15	37,5	0,010

TABLE VI
SW CONFIGURATION SIZE

Design	Used MULs	Used LUTs	Used FFs	Used Slices	% Slot Slices
uCDPr*	0	2	18	9	4,6
HW_AVRF**	0	253	306	273	35,5
HW_DSACC_v1	0	170	68	311	54
HW_DSTMP_v1	0	71	40	127	22
HW_DSACC_v2	2	127	68	232	40
HW_DSTMP_v2	2	63	40	111	19

*Slot 0, **Slot 1

The FPGA area needed by some hard cores are presented in Table VI: i) for the μ C interfaces (slot 0), the digital version is included (*uCDPr*) as the analog one does not consume FPGA LUTs; ii) a moving average coprocessor filter (*HW_AVRF*) is included; and iii) for the sensor interfaces, one Freq/Period digital temperature sensor (*HW_DSTMP*) and one digital PWM-based accelerometer (*HW_DSACC*). Notice that there are two versions of each interface hard core: one that uses the FPGA embedded multiplier (_v2) and one that does not uses them (_v1). The table also includes the percentage usage of the corresponding slot.

As it can be noticed from Table VI, it is not possible to have all the interfaces and data processing loaded in the FPGA, that is another advantage of using partial reconfiguration.

To define an application on top of a partial runtime reconfigurable system, the first step (before any partial reconfiguration) is to perform an FPGA full configuration. In the approach followed in this paper, the first FPGA configuration is used to load the virtual architecture definition, that is, to reserve slot areas and to load the on-chip communication resources. This configuration is kept in the FPGA boot-loading memory, in a separate device from where the configuration is read on power up (in the new Spartan3AN FPGA this memory is embedded in the silicon die). In the initial configuration, feedthrough configurations are loaded in slot 0 and slot 1, while slot 2 is left empty. After this, any application can be arranged on the FPGA. Two applications, described next, have been selected as a use case.

1) Analog sensors with HW average filtering (AS_HW_AVRF). For building this application, a feedthrough configuration is needed in slot 0, in slots 1 the average filter hard core and finally slot 2 has to be left empty. The physical implementation of this application in the FPGA can be seen on the left side of Fig. 6, where a screenshot of the FPGA Editor tool (integrated in ISE) is presented.

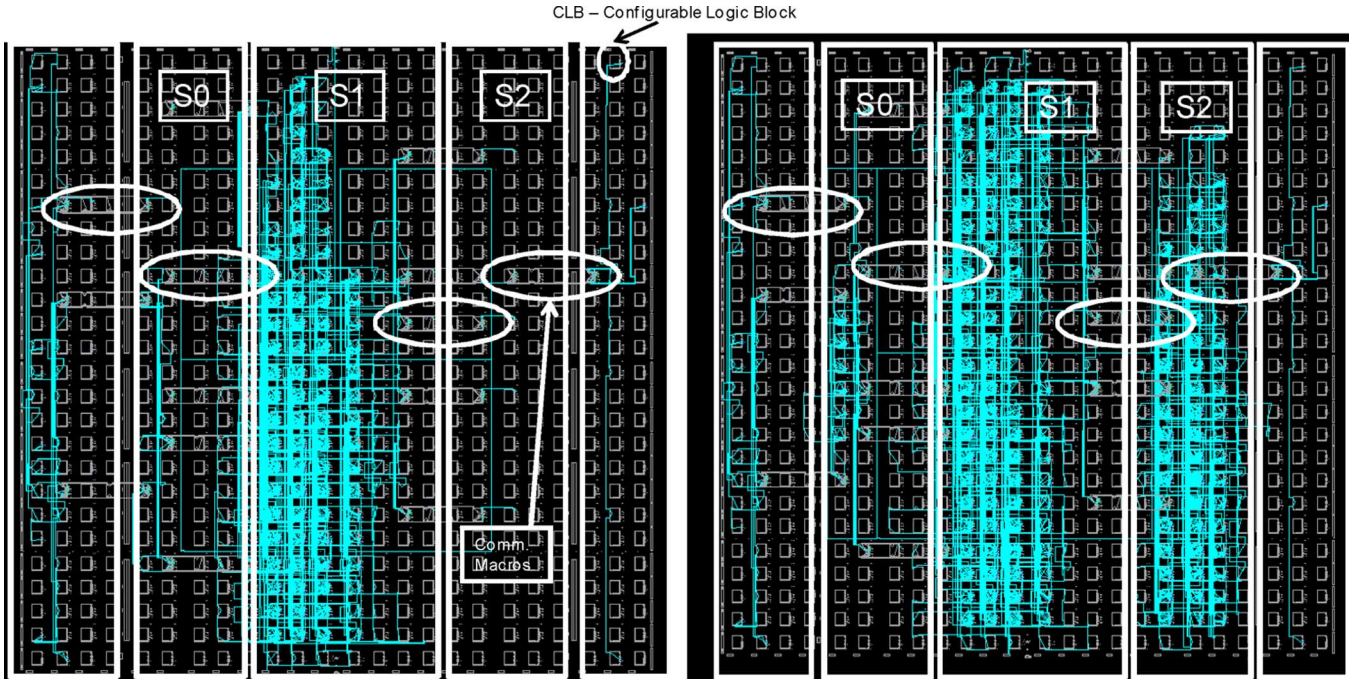

Fig. 6. FPGA Editor tool screenshots that shows two system applications. An analog sensor with average filtering (AS_HW_AVRF) is shown in the figure on the left and a digital accelerometer sensor with the same average filtering (DS_HW_AVRF_ACCIF) in the figure on the right. The partial reconfigurable system slots can be noticed: slot 0 defines the uC-to-FPGA interface, slot 1 is reserved data processing and slot 2 allocates digital sensor interfaces.

2) Digital accelerometer sensor with HW average filtering (DS_HW_AVRF_ACCIF). For building it, an uCDPr configuration is needed in slot 0, in slots 1 the average filter hard core and finally the temperature sensor interface has to be loaded in slot 2. The physical implementation of this application can be seen on the right side of Fig. 6.

In order to validate the runtime reconfigurable system a sequence of partial runtime reconfigurations have been performed, while the NUR μC is kept sending data to a host system where it is printed out on a terminal console (the μC reprogramming has also been tested, but it has not been included in this paper). After the mentioned initial full configuration, several runtime reconfigurations, listed next, have been executed.

1) The average filter HW configuration has been sent to the NUR and loaded in slot 2 to pass from the empty configuration to the AS_HW_AVRF application. This reconfiguration has a cost of $\mathrm{Ctotalreconf} = 0\,0054$.
2) Slot 0 and slot 2 have been reconfigured to pass from AS_HW_AVRF application to the DS_HW_AVRF_ACCIF application with a total cost of $\mathrm{Ctotalreconf} = 0\,0054$.

While performing these reconfigurations, no disruption or errors in system data transitions have been noticed, thus the remote partial runtime reconfiguration tests have been successful.

VI. Summary and Discussion

In the previous sections, a runtime reconfigurable solution for WSNs has been proposed and evaluated. The selected approach, apart from the μC reprogramming, provides a solution that exploits partial runtime reconfiguration capabilities of the FPGA included in the Cookie, in order to increase flexibility, reduce the amount of data to be transmitted and reduce the reconfiguration time. The provided solution permits to modify at runtime: i) the sensor HW interfaces; ii) the coprocessor allocated in the FPGA for taking advantage of the HW parallelism; and ii) the μC-to-FPGA interface. Also, a use case has been set up and remote runtime partial reconfiguration in WSNs nodes has been validated. Furthermore, the presented system can be ported to other FPGAs and also to other virtual architectures, except from the hard core library that has to be regenerated from the source code.

Nevertheless, there are still some open points that may be discussed in this field, mainly related to the FPGA. A standard design decision in reconfigurable systems is the FPGA to μC coupling where several options are available: from standalone systems where the FPGA and the μC are separate devices (like in the system presented in this paper), to tightly coupled systems where the μC is embedded in the FPGA (either as soft-core implemented with FPGA logic resources or melt in the silicon die). The selection of one or another option depends on the specific application requirements. In the work presented here, the node standalone coupling has been maintained, instead of embedding a soft-core μC in the FPGA, because the main goal was to create and test the runtime reconfigurable system. The target Spartan 3 FPGA belongs to the Xilinx low cost, small, device series. In this device, reserving area to allocate the μC has the effect of high reduction of the number of reconfigurable slots and thus system reconfiguration flexibility.

Other important aspects related to the use of FPGAs are the power consumption (and thus, their ability to be used in WSNs), configuration time, security of reprogramming actions, etc. However, advances in electronic and communication technologies will accommodate the efficient use of reconfigurable

hardware in commercial WSNs. As an example: low-power FPGAs that consume less than 5 μW in standby mode (Actel IGLOO FPGA, mentioned in Section II) are currently available. Furthermore, state-of-the-art FPGAs are constantly increasing the diversity of embedded resources. Now, there are high performance FPGAs that include up to four microprocessor and specialized signal processing logic in the die (Xilinx Virtex 5 for instance) and others that include ADCs and PWMs (Actel Fusion FPGA for instance). As a result, it could be said that future highly reconfigurable devices have a place in WSN industry.

VII. Conclusion

The work presented in this paper is an approach to open new horizons to the use and development of runtime reconfigurable Wireless Sensor Networks. The work shows the feasibility of the integration of new reconfiguration techniques in WSN that may lead to change the use of WSNs, afrom custom made for a given application, to generic networks deployments where tasks and functions are dynamically allocated. The considered updates cover both runtime HW reconfiguration and SW updates, and may include new functionalities, updating of security information, dynamic task allocation, online testing, debugging and repair, among other features.

The integration of reconfigurability in WSN nodes might result in alleviating some costly activities in the industrial use of WSNs, like deployment and maintenance.

References

[1] L. Q. Zhuang, K. M. Goh, and J. B. Zhang, "The wireless sensor networks for factory automation: Issues and challenges," in *Proc. IEEE Conf. Emerging Technol. Factory Autom., ETFA'07*, Sep. 25–28, 2007, pp. 141–148.

[2] A. Willig, "Recent and emerging topics in wireless industrial communications: A selection," *IEEE Trans. Ind. Informat.*, vol. 4, no. 2, pp. 102–124, May 2008.

[3] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *Proc. 4th Int. Symp. Information Processing in Sensor Networks, IPSN'05*, Apr. 2005, pp. 364–369.

[4] S. Yamashita, T. Shimura, K. Aiki, K. Ara, Y. Ogata, I. Shimokawa, T. Tanaka, H. Kuriyama, K. Shimada, and K. Yano, "A 15 × 15, 1 μA, reliable sensor-net module: Enabling application-specific nodes," in *Proc. 5th IEEE/ACM Int. Conf. Inform. Process. Sensor Networks, IPSN'06*, Apr. 2006, pp. 383–390.

[5] J. J. Rodrigez-Andina, M. J. Moure, and M. D. Valdes, "Features, design tools, and application domain of FPGAs," *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1810–1823, Aug. 2007.

[6] W. H. Mangione-Smith *et al.*, "Seeking solutions in configurable computing," *IEEE Computer*, vol. 30, no. 12, pp. 38–43, Dec. 1997.

[7] R. Scrofano, M. B. Gokhale, F. Trouw, V. K. E. Monmasson, and M. N. Cirstea, "FPGA design methodology for industrial control systems—A review," *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1824–1842, Aug. 2007.

[8] H. Chaoui, M. C. E. Yagoub, and P. Sicard, "FPGA implementation of a fuzzy controller for neural network based adaptive control of a flexible joint with hard nonlinearities," in *Proc. IEEE Int. Symp. Ind. Electron.*, Jul. 2006, vol. 4, pp. 3124–3129.

[9] R. Scrofano, M. B. Gokhale, F. Trouw, and V. K. Prasanna, "Accelerating molecular dynamics simulations with reconfigurable computers," *IEEE Trans. Parallel and Distrib. Syst.*, vol. 19, no. 6, pp. 764–778, Jun. 2008.

[10] H. Hinkelmann, P. Zipf, and M. Glesner, "Design concepts for a dynamically reconfigurable wireless sensor node," in *Proc. 1st NASA/ESA Conf. Adaptive Hardware and Systems, AHS'06*, Jun. 2006, pp. 436–441.

[11] A. E. Susu, M. Magno, A. Acquaviva, and D. Atienza, "Reconfiguration strategies for environmentally powered devices: Theoretical analysis and experimental validation," *Trans. HiPEAC I, LNCS 4050*, pp. 341–360, 2007.

[12] J. Becker *et al.*, "Dynamic and partial FPGA exploitation," *Proc. IEEE*, vol. 92, no. 2, pp. 438–452, Feb. 2007.

[13] P. Sedcole, Y. K. Peter, C. George, A. Constantinides, and W. Luk, "Run-time integration of reconfigurable video processing systems," *IEEE Trans. VLSI Syst.*, vol. 15, no. 9, pp. 1003–1016, Sep. 2007.

[14] J. Portilla, A. de Castro, E. de la Torre, and T. Riesgo, "A modular architecture for nodes in wireless sensor networks," *J. Univ. Comput. Sci. (JUCS)*, vol. 12, no. 3, pp. 328–339, Mar. 2006.

[15] *IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*, IEEE Standard 802.15.4-2003, 2003, pp. 1–670.

[16] D. Lymberopoulos, N. B. Priyantha, and F. Zhao, "mPlatform: A reconfigurable architecture and efficient data sharing mechanism for modular sensor nodes," in *Proc. 5th IEEE/ACM Int. Conf. Inform. Process. Sensor Networks, IPSN'07*, Apr. 2007, pp. 128–137.

[17] B. O'Flynn, S. Bellis, K. Delaney, J. Barton, S. C. O'Mathuna, A. M. Barroso, J. Benson, U. Roedig, and C. Sreenan, "The development of a novel minaturized modular platform for wireless sensor networks," in *Proc. 4th Int. Symp. Inform. Process. Sensor Networks, IPSN'05*, Apr. 2005, pp. 370–375.

[18] *IEEE Standard Test Access Port and Boundary-Scan Architecture*, IEEE Standard 1149.1-2001, 2001, pp. i–200.

[19] DS1WM Synthesizable 1-Wire Bus Master Datasheet, Maxim Integrated Products.

[20] J. Portilla, J. L. Buron, A. de Castro, and T. Riesgo, "A hardware library for sensors/actuators interfaces in sensor networks," in *Proc. IEEE Int. Conf. Electron., Circuits and Systems 2006 (ICECS'06)*, Niza, France, Dec. 2006, pp. 1244–1247.

[21] C. Steiger, H. Walder, and M. Platzner, "Operating systems for reconfigurable embedded platforms: Online scheduling of real-time tasks," *IEEE Trans. Comput.*, vol. 53, no. 11, pp. 1393–1407, Nov. 2004.

[22] V. Nolet, P. Avasare, H. Eeckhaut, D. Verkest, and H. Corporaal, "Run-time management of a MPSoC containing FPGA fabric tiles," *IEEE Trans. VLSI Syst.*, vol. 16, no. 1, pp. 24–33, Jan. 2008.

[23] K. Paulsson, M. Hubner, G. Auer, M. Dreschmann, L. Chen, and J. Becker, "Implementation of a virtual internal configuration access port (JCAP) for enabling partial self-reconfiguration on Xilinx Spartan-III FPGAs," in *Proc. 17th IEEE Int. Conf. Field Programmable Logic, FPL'07*, Aug. 2007, pp. 351–356.

[24] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght, "Modular dynamic," *Proc. IEE Comput. Digit. Tech.*, vol. 153, no. 3, pp. 157–164, May 2006.

[25] Blank for Blind Review.

[26] D. Lim and M. Peattie, Xilinx, "XAPP 290 (v1.0): Two flows for partial reconfigurtion: Module based or small bit manipulation," *XAPP 290*, 2004.

[27] E. L. Horta and J. W. Lockwood, "Automated method to generate bitstream intellectual property cores for Virtex FPGAs," *Proc. 14th Field-Programmable Logic and Applications, FPL04*, pp. 975–979, Aug. 2004.

[28] Y. E. Krasteva, J. Portilla, J. M. Carnicer, E. de la Torre, and T. Riesgo, "Wireless sensor networks node with remote HW/SW reconfiguration capabilities," in *Proc. IEEE Annu. Conf. IEEE Ind. Electron. Soc. (IECON'08)*, Orlando, FL, Nov. 2008, pp. 2483–2488.

[29] B. Bridgford and J. Cammon, Xilinx, "SVF and XSVF file formats for Xilinx devices," *XAPP 503*, 2007.

Yana Esteves Krasteva was born in Lima, Perú, in 1979. She received the M.Sc. degree from the Technical University of Sofia, Sofia, Bulgaria, in 2002 and the Ph.D. degree in electronic engineering from the Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 2009.

Since 2010, she has been a Researcher in the Parallel Architectures Group (GAP), Universidad Politécnica de Valencia (UPV). She has worked in several research projects with European and national funding, as well in projects for the industry. Her research interests are focused on supercomputing and reconfigurable systems, networks on chip and wireless-sensor network.

Jorge Portilla was born in Madrid, Spain, in 1978. He received the M.Sc. degree in physics from the Universidad Complutense de Madrid (UCM), Madrid, Spain, in 2003 and the Ph.D. degree in electronic engineering from the Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 2010.

He is an Assistant Professor since 2006 at the Universidad Politécnica de Madrid, and doing his research at the Industrial Electronics Centre (CEI). His research interests are focused mainly in wireless sensor networks, reconfigurable systems, and embedded systems design.

Eduardo de la Torre was born in Madrid, Spain, in 1964 and received the M.Sc. degree in industrial engineering in 1989 and the Ph.D. degree in industrial electronics in 2000 from the Universidad Politécnica de Madrid, Madrid, Spain.

He is an Associate Professor at Universidad Politécnica de Madrid, and doing his research at the Industrial Electronics Centre. He has worked in integrated circuit design for ASICs and FPGAs, mostly for control applications, as well as embedded systems design. He has also worked in tools for test and debug of these systems. Recently, his main research interests are reconfigurable systems and networks on chip.

Teresa Riesgo was born in Madrid, Spain, in 1965. She received the M.Sc. and Ph.D. degrees in electrical engineering from Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 1989 and 1996, respectively.

Since 2003, she is a Full Professor of Electronics at UPM. She is currently the Director of the Center of Industrial Electronics of Universidad Politécnica de Madrid (CEI-UPM). She has published a large number of papers in these fields and has participated and acted as main researcher in several European Union-funded projects. Her research interests are focused on embedded system design, wireless-sensor networks, configurable systems, and power estimation in digital systems.