Remote HW-SW Reconfigurable Wireless Sensor Nodes

Y. E. Krasteva, J. Portilla, J. M. Carnicer, E. de la Torre, T. Riesgo
Centro de Electronica Industrial, Universidad Politecnica de Madrid
{yana.ekrasteva, jorge.portilla, eduardo.delatorre, teresa.riesgo}@upm.es

***Abstract*- Reconfigurable HW, like FPGAs, can improve the processing systems performance as it has been demonstrated by several research groups. Usually, the inclusion of such elements in HW platforms for Wireless Sensor Networks (WSNs) has been rejected by designers, mainly due to the power consumption penalization. A reconfigurable device allows not only performance improvement but also remote HW reconfiguration of the WSN node. In this paper, a entire working flow for generate, remotely configure and reconfigure the HW in a target custom reconfigurable platform developed at CEI (Centro de Electronica Industrial) is presented. The custom platform includes a microprocessor and an FPGA (Xilinx partially reconfigurable) to carry out all the processing tasks. The current reconfiguration process works with the JTAG interface, which makes the solution portable to other FPGAs, especially those new less power consuming devices that are appearing in the market nowadays.**

I. INTRODUCTION

Nowadays Wireless Sensor Networks represents one of the most outstanding and ambitious challenges in electronic design and telecommunications. These special networks are intended to be autonomous, low power, context aware and reconfigurable.

The particular requirements related to WSNs, like low power consumption and high performance, force designers to think in a new way. In this context, the node hardware becomes a critical issue.

Typical applications for WSNs may include hundreds or thousands of nodes. In such situations, it is very important to be able to debug the network and reprogram the nodes on-line, to solve problems of working or performance. These tasks are related to the commissioning concept [1].

Usually, WSNs HW designers include a microcontroller (uC) as the smart element, which carries out all the tasks related to communication control, sensors processing and global node management [2][3]. At CEI, a modular HW reconfigurable platform has been developed during the last years, called Cookie. This platform includes a uC and an FPGA as processing elements. The idea is to have a processing support for the uC, to carry out specific tasks which could overload the uC (special processing as video and audio or sensor processing for particular digital interfaces, among others). HW acceleration for WSNs has been also targeted by other groups as [4][5], but their approach includes the development of specific reconfigurable integrated circuits. Differently, Cookies include commercial FPGAs. Therefore the design process is based also on commercial, well known tools, which reduces the time to market.

One of the problems that arises when a reconfigurable element as an FPGA is introduced in a system like a WSN node is the power consumption. Commonly used FPGAs have high power consumption, like the currently used in the Cookies, provided by Xilinx. But low power FPGAs are appearing in the market like the Actel Igloo [6], that soon will substitute the Xilinx one in Cookies.

HW reconfiguration features improve node flexibility and computation performance, and opens the possibility of remote HW reconfiguration, which can be very useful to tune node performance with new actualizations, to debug on-line (commissioning) the WSN, and make the node smarter.

This paper details the work developed to remotely reconfigure the FPGA HW, integrated in the node, using a wireless link (ZigBee in this case). An application example in which this node HW reconfiguration capability is intended to be used is presented as well.

The basic idea is to use the uC to reconfigure the FPGA. The uC will use the JTAG port of the FPGA to accomplish this task. This decision was taken because most of the market FPGAs include a JTAG port to download its configuration file. The uC will receive the bitstream from the ZigBee module and will download it in the FPGA configuration memory.

The current node design includes a Xilinx XC3S200 Spartan 3. This work has taken advantage of the reconfigurability research line developed at CEI and briefly overviewed in section II, focused on exploring partial reconfigurable capabilities of these devices (other manufactures do not offer this feature). Moreover, with this approach the size of bitstreams are much lower, and the remote configuration process is much faster.

The idea will be tested with hardware sensor interfaces reconfiguration. The main concept is to have a library of general HW digital interfaces for sensors (as I2C, 1-Wire, SPI, etc.). These interfaces are designed in a general way in order to allow being adapted to new sensors in a fast way. The demonstrator will start with a default configuration for the FPGA with some interfaces configured and one of these interfaces will be replaced by another using dynamic partial reconfiguration. Partial reconfiguration not only permits to modify only part of the FPGA configuration without affecting the rest of the logic loaded in it, but also without the need of restarting or stopping the node.

The rest of the paper is organized as follows. Section II details the HW node architecture and the HW sensor interfaces. Section III describes the remote reconfiguration process and section IV how the partial reconfiguration is carried out. Section V details the proof of concept and finally section VI concludes the paper.

II. COOKIE SENSOR NODE

The WSN node in which reconfiguration tasks are carried out is the Cookie [7]. (see "Fig. 1"). This node was designed with a main philosophy in mind: modularity.

Modularity allows dividing and encapsulating the functionalities included in the node. Therefore, future redesigns may involve only part of the platform, which is desirable considering the time to market. Moreover, researching using this platform turns open, due to the node flexibility, which makes possible the proof of several concepts minimizing the effort.

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

1. Processing: This is the heart of the node. It includes a 8052 uC from Analog Devices (ADuC841) and a Xilinx XC3S200 Spartan 3 FPGA. The uC and the FPGA share 3 8-bit ports for communication. Moreover, the JTAG port of the FPGA is connected to the uC to makepossible remote reconfiguration.

2. Communication: The last version of this layer includes a ZigBee module (ETRX2 from Telegesis). This module is controlled by the uC through the UART port. A layer version with Bluetooth is also available, controlled through the UART as well, keeping modularity.

3. Power supply: This layer generates all the voltages needed within the Cookie. Two versions have been developed, the latest with USB included, which allows power supply from a PC and serial programming for the uC.

4. Sensors: This layer includes those elements which are intended to take measures from the environment. Today, 3 different layers have been developed for the Cookies. These layers include sensors of acceleration, temperature, humidity, light, infrared and deformation.

One of the features of the sensor layer is that it can include sensors with both analog and digital interfaces (they will be called analog sensors and digital sensors). Signals from analog sensors are connected to the ADC of the uC. On the other hand, signals from digital sensors are connected to the FPGA. In principle, the FPGA carries out all the processing related to digital sensors, to release the uC, which manage the communications and processes analog sensors.

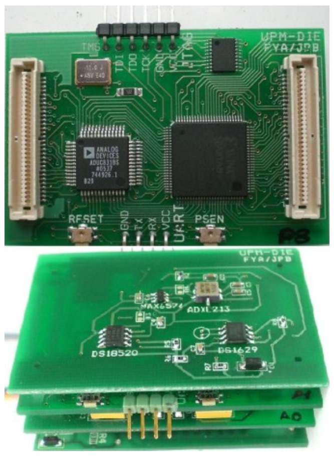
Fig. 1 Processing layer (uC on the left, FPGA on the right) and full Cookie platform

Nowadays, there are a myriad of sensors in the market with several different interfaces to communicate their measurements. Many of them include digital interfaces, with different protocols as SPI, I2C, 1-Wire, etc. When this kind of signals have to be processed using a uC, problems related to timing and processor overhead can appear. In fact, some manufacturers offer HDL code to implement the sensor interfaces in a coprocessor [8].

In this context, a library of general HW interfaces [9] has been developed at CEI in order to process signals of sensors with very different digital interfaces. The interfaces included in the library until now are:

- I2C and I2C modified (Sensirion company interface)
- PWM
- Period/Frequency
- 1-Wire

The library is divided in modules which represent sensor (or actuator) interfaces. Every module has been designed following a philosophy inspired in the IEEE 1451 family of standards, but can also be used without being compatible with them. Each transducer is "seen" as a channel (or set of channels) by the sensor or actuator (transducer) controller. Two kinds of channels 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 uC sends triggers to the FPGA, specifying the sensor from which the measure has to be taken, and the FPGA activates the corresponding sensor interface. Finally, the FPGA sends results of the measure in two bytes. So, the FPGA acts as a reconfigurable coprocessor for the uC

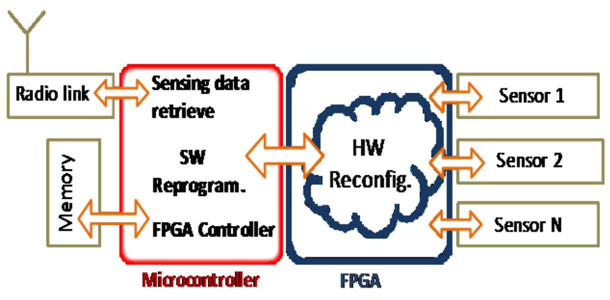

Fig. 2. HW-SW Reconfigurable Sensor Node Diagram

III. Remote Reconfiguration

The remote reconfigurable system schematic view is shown in "Fig. 2". The remote reconfiguration problem in the Cookie platform presents two aspects to be covered: uC reprogramming and FPGA reconfiguration.

A. *uC Reprogramming*

The uC included in the Cookie (ADuC841 from Analog Devices) is programmed using its UART port. The uC includes a serial protocol in order to be able to program the uC through a host (i. e. a PC or other processor).

On the other hand, the ZigBee module is connected to the uC serial port. The uC sends commands to the ZigBee module to manage the communications and sends the raw data to be delivered to the network. In order to reprogram the uC using the wireless ZigBee link a manager program (called Cookie Manager) has been developed. This manager carries out all the steps related to uC program downloading through the serial port. First of all, a ZigBee channel must be establish between the sink (network coordinator) and the remote node in which the reprogramming has to be done. Next, Cookie Manager sends all the commands needed to program the remote uC as if a real serial cable was connected between the host programmer and the remote node. Finally, the programming file is sent and is introduced in the uC flash memory. The serial downloading protocol allows starting code executing in the remote uC through a specific command.

Not only a program but other data can be loaded into the uC flash memory. So a typical file downloaded to the uC to configure the FPGA includes a program that acts as a JTAG controller and a file which includes the bitstream to be loaded into the FPGA with the instructions that the JTAG controller has to execute to carry out the reconfiguration.

B. *FPGA Reconfiguration*

Regarding the FPGA reconfiguration, the selected Spartan 3 does not have an internal configuration port (ICAP), therefore the remaining configuration port options are: Select Map and JTAG. As it has been already mentioned, JTAG is the selected reconfiguration interface. A SW implementation of the JTAG controller, provided by Xilinx, has been used [12]. Therefore no additional FPGA area is required for the reconfiguration process control, like for instance if Select Map was selected, like in [10]. On the other hand, the main disadvantage of using JTAG is the reconfiguration time (JTAG is serial, while Select Map parallel).

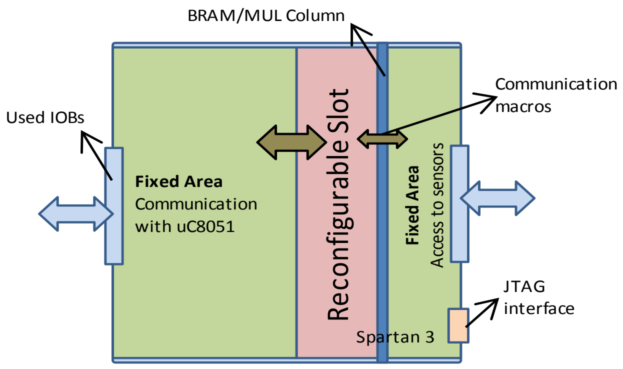

Fig. 3. HW Reconfigurable Sensor Node Spartan 3 FPGA Virtual Architecture

The JTAG configuration program running on the uC uses Xilinx specific Boundary Scan configuration files (.xsvf) where the configuration data is formatted in binary commands. In the present paper, only partial bitstreams are loaded into the FPGA, taking advantage of the partial reconfiguration capabilities of the Xilinx Spartan 3 FPGA.

To retarget the system to other FPGAs (non Xilinx) that can be programmed by JTAG, a Xilinx tool called svf2xsvf [11] can be used to transform a boundary scan standard ASCII file to a Xilinx binary file.

Once the FPGA has been configured by the uC, the final code for the application is downloaded into the uC flash memory. This step is done in the same way as it was explained before.

All these tasks can be carried out between any two Cookies, exploiting the multihop capabilities of the ZigBee standard. This allows reconfiguring every node connected to the network with the only drawback of time delay. Tests have been done for a two hop configuration as a proof of concept. These results are shown in section V.

IV. FPGA Partial Reconfiguration

Partial reconfiguration in FPGAs requires defining *Virtual Architectures (VA)* on top of the selected device. For this aim, a set of steps have to be followed. Details of VAs design are outside the scope of this paper [13]. Here, only the main aspects applied to a target Spartan 3 FPGA are included.

The first aspect to be taken into account is the VA model. VA models can be one dimensional (1D) or two dimensional (2D). According to [14], partial reconfiguration in Virtex II based architectures (this includes Spartan 3 FPGAs) is frame based, i.o.w. only reconfiguration regions that span entire FPGA columns can be defined and this permits a direct use only of 1D based VAs. A general view of the defined VA for the sensor node Spartan 3 FPGA is presented in "Fig. 3".

The structure of the Spartan 3 FPGA is quite regular in comparison with other FPGAs (Virtex II for instance). Therefore, the fixed area (the one that never changes) of the FPGA occupies only the left and right FPGA sides, because of the mentioned regularity. The right side is in charge of the communication with the external microcontroller, while the left one is used to access the node sensors. The remaining of the FPGA, defined as reconfigurable area has a very regular structure and is divided into reconfigurable *Slots* (slots are composed only of CLB columns). In this, first approach, only one slot has been defined in the reconfigurable area because it is restricted by the needed Input Output Blocks (IOBs) for the communication with the microcontroller (right fixed area). The defined slot is 6 CLBs wide and is composed of 144 CLBs (576 Slices) in the target XC3S200 FPGA.

For slot communication with fixed areas, structures called Bus Macros are used. Two types of bus macros have been designed. In this implementation, first, a horizontal bidirectional macro that passes four bits of data from left to right and four bits from right to left. This macro is similar to the Xilinx Bus Macros, available for Spartan 3 in [15], but the difference is that these ones are bidirectional. Our experience shows that better routing can be achieved using bidirectional macros instead of unidirectional. The second type is also a bidirectional macro, but it is used to cross the BRAM/MUL column that occupies the right fixed area to slot border. This BRAM/MUL column can be accessed by modules (HW Cores) loaded into the slot.

The working flow that has been followed is presented in "Fig. 4". Once the FPGA VA has been defined, that includes a user constraint file and the set of communication macros to be used, the conventional (ISE) design flow can be followed. This flow ends with the generation of a full configuration file, from which the slot configuration region is extracted using *bitgen* (the Xilinx tool for bitstream generation) with partial mask options. Another approach is the use of the Xilinx partial reconfiguration design flow, based on ISE and the *PlanAhead* tool (suitable for testing different floorplanning approaches). Differently from the first approach, this one directly results in partial configuration files and has better routing. Anyway, in the current implementation we have selected the first one because we have plenty of space in the slot and the virtual architecture floorplanning has been well defined in the first step of the working flow.

Independently from the method used to generate partial configuration files, the next step is to use the iMPACT tool to generate the needed binary formatted boundary scan configuration file to partially reconfigure the node FPGA.

New FPGA partial configurations as well as new node microprocessor configurations are sent using the previously described custom software (Cookie Manager).

V. Use Case and Results

A use case has been setup as a proof of concept. Two different applications have been created on top of the system presented in this paper and have been alternatively loaded by remotely reconfiguring and reprogramming the sensor node HW and SW.

The first application uses the node accelometer and is refereed as *ACCS,* while the second one, referred as *TMPS*, uses the node temperature sensor. The applications HW part, mapped in the FPGA slot along with all the virtual architecture components can be seen in "Fig.5". Both designs

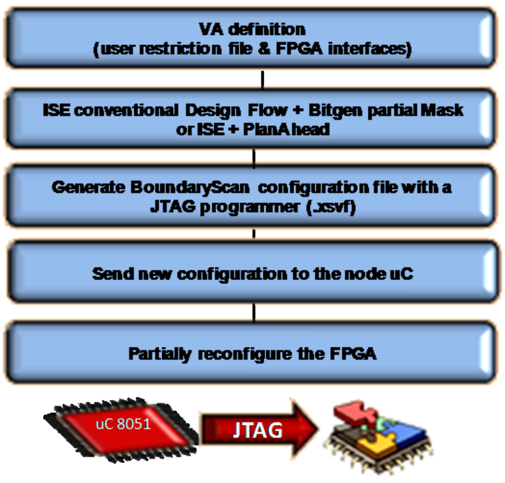
VA definition
(user restriction file & FPGA interfaces)
ISE conventional Design Flow + Bitgen partial Mask or ISE + PlanAhead
Generate BoundaryScan configuration file with a JTAG programmer (.xsvf)
Send new configuration to the node uC
Partially reconfigure the FPGA
uC 8051
JTAG

Fig. 4. Working Flow

HW parts include multiplier functions that can be implemented either using the embedded multipliers of the FPGA or simply using FPGA LUTs. The used area by the HW part of each application and each application version (with and without embedded multipliers) is presented in Table I. The percentage of the used slot area is also included, and it can be noticed that the slot is underused and therefore bigger designs can be loaded in it without having to redefine the FPGA VA.

TABLE I
FPGA Used Area

Design	Used MULs	Used Slices	Used FFs	Used LUTs	% of Slot Slices
ACCS_HW_v1	0	170	68	311	29
TMPS_HW_v1	0	71	40	127	12
ACCS_HW_v2	2	127	68	232	22
TMPSI_HW_v2	2	63	40	111	10

Partial configuration files have been generated from each design using the working flow described in the previous section and the resulting file sizes in both .bit and .xsvf format are presented inTable II. Since the reconfiguration unit is a slot, the configuration files sizes are the same for both applications in "XX_v1" (without using the embedded MULs). For "XX_v2", with MULs, designs use not only the slot, but also the standing next to it BRAM/MUL column and therefore file size increase considerably. As reference it is important to mention that a complete XC3S200 configuration file is 131 KB and thus makes partial reconfiguration very useful for resource restricted devices.

The microcontroller program memory is 62 KB. New HW and SW configurations are loaded in this memory. The JTAG configuration program occupies 6K of the available memory, while the rest is left for the HW configuration files. A set of HW/SW configurations for each application (ACCS and TMPS) has been sent to the reconfigurable node using different type of transmission media: i) using the available communication in the sensor node (a ZigBee module) and ii)

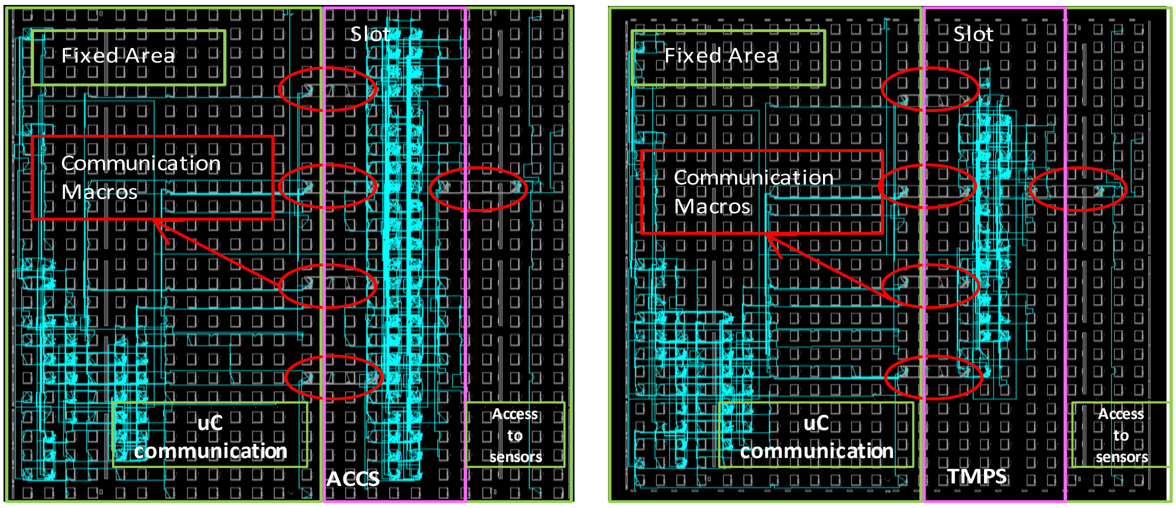

Fig. 5. System Applications: ACCS_HW on the left side and TMPS_HW on the right side

using a serial cable connection. Table III summarizes both transmission times and data rates.

TABLE II
PARTIAL CONFIGURATION FILES SIZE

Design	.bit (KB)	.xsvf (KB)
ACCS_HW_v1	24,9	25,2
TMPS_HW_v1	24,9	25,2
ACCS_HW_v2	41,3	43,5
TMPSI_HW_v2	41,3	43,5

In the table two different packetized formats can be differentiate: 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 *Node Under Reconfiguration* (NUR) and ii) in a long distance connection with the NUR, using a node for intermediate routing (multihop). Results, included in Table III, have been measured under direct connection mode.

TABLE III
CONFIGURATION TIMES

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

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 transmission fails with a high rate in long distance connections compared with 8 bytes based transmissions that are much more reliable, but take considerably longer time. It is well known that the ZigBee standard is intended for transmitting small amount of data, also, the currently used ZigBee transmitter module does not permit low level protocol optimizations and therefore, the presented results are far from being optimal. Anyway, with the expansion of the wireless sensor networks application range, new standards for nodes communication appear, like a low power WiFi version [16].

The final step, once the HW and SW configuration have been loaded into the uC program memory, is to execute the JTAG programmer. The time needed for this application to partially program the FPGA is 4.2 seconds. As a comparison the time needed by the Xilinx iMPACT SW running on a PC is one second. This time can be drastically reduced if a special, FPGA specific, parallel programmer is used.

VI. CONCLUSIONS AND FUTURE WORK

The presented wireless sensor network node permits SW and/or HW updates. New configurations are transmitted using the wireless network. The selected approach exploits partial reconfiguration capabilities of the node Spartan 3 FPGA. This permits to reduce the amount of data to be transmitted (configuration files are smaller). Since the reconfiguration process is base on the JTAG standard, the solution is portable to other FPGAs.

Remote reconfiguration provides the possibility of building intelligent nodes, while the FPGA gives higher performance and flexibility. Such nodes can be used to build dynamically adaptable WSNs that can be modified to achieve a certain goal even after deployment. A use case has been set up to evaluate the possibility of using the network for new configuration transmissions. Results show that the selected wireless communication module, based on the ZigBee standard, is not suitable. High delay and fail rates during the reconfiguration process have been reported. Anyway, this problem can be easily solved by simple changing the communication layer of the modular platform with a new, updated one, based on more reliable and fast communication, like WiFi. Such layer design has already been planned in the near future work. Additionally, the main drawback of the currently used Xilinx FPGAs is the high power consumption. Again taking advantage of the modularity of out platform, a new processing layer is under development that includes a very low power, full reconfigurable, FPGA (5 uW best case consumption) provided by Actel.

REFERENCES

[1] Analysis of Deployment Methodologies for Wireless Sensor Networks, FP6-2005-IST-034642 European Project, Deliverable 8.

[2] S. Yamshita, T. Shimura, K. Aiki, K. Ara, Y. Ogata, I. Shimokawa, T. Tanaka, H. Kuriyama, K. Shimada, K. Yano, "A 15x15, 1 μA, Reliable Sensor-Net Module: Enabling Application-Specific Nodes," *in Proc. of the 5th IEEE/ACM International Conference on Information Processing in Sensor Networks (IPSN'06)*, April 2006, pp. 383–390.

[3] J. Polastre, S. Szewczyk, D. Culler, "Telos**:** Enabling Ultra-Low Power Wireless Research", *in Proc. of the 4th IEEE/ACM International Conference on Information Processing in Sensor Networks (IPSN'05),* April 2006, pp. 364-369.

[4] H. Hinkelmann, P. Zipf, M. Glesner, "Design Concepts for a Dinamically Reconfigurable Wireless Sensor Node", *in Proceedings of the 1st NASA/ESA Conference on Adaptive Hardware and Systems (AHS'06)*, pp. 436-441, Jun 2006.

[5] A. E. Susu, M. Magno, A. Acquaviva, D. Atienza, "Reconfiguration Strategies for Environmentally Powered Devices: Theoretical Analysis and Experimental Validation", *in Transactions on HiPEAC I, LNCS 4050*, pp. 341-360, 2007.

[6] http://www.actel.com/products/IGLOO/default.aspx

[7] J. Portilla, A. de Castro, E. de la Torre, T. Riesgo, "A Modular Architecture for Nodes in Wireless Sensor Networks", *Journal of Universal Computer Science (JUCS)*, vol. 12, nº 3, March 2006, pp. 328-339.

[8] "DS1WM, Synthesizable Verilog 1-Wire Bus Master", Dallas - Maxim, http://www.maxim-ic.com/.

[9] J. Portilla, J.L. Buron, A. de Castro, T. Riesgo, "A Hardware Library for Sensors/Actuators Interfaces in Sensor Networks", *Proc. of the 13th IEEE International Conference on Circuits and Systems*, Nice, France, Dec. 2006.

[10] 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 proceedings of 17th IEEE international conference on Field Programmable Logic* (FPL07), Amsterdam, Netherland, August 2007.

[11] Brendan Bridgford and Justin Cammon, "SVF and XSVF File Formats for Xilinx Devices ", XAPP 503, Xilinx, 2007.

[12] "XAPP 058 (v. 4.0): Xilinx In System Programming Using an Embedded Microcontroller", October 2007.

[13] Yana E. Krasteva, Eduardo de la Torre, Teresa Riesgo, "Virtual Architectures for Partial Runtime Reconfigurable Systems. Application to Network on Chip based SoC Emulation", *to be published in Proc. of the The 34th Annual Conference of the IEEE Industrial Electronics Society (IECON 08),* Nov. 2008

[14] Davin Lim, Mike Peattie, "XAPP 290 (v1.0): Two Flows for Partial Reconfigurtion: Module Based or Small Bit Manipulation", *XAPP 290,Xilinx,* 2004.

[15] "Partial Reconfiguration Software User's Guide", *Xilinx*, 2006.

[16] http://www.gainspan.com/.