Alvaro M. AVELINO[1], Valentin O. RODA[2], Carlos A. V. SAKUYAMA[3], Glauberto A. L. DE ALBUQUERQUE[3], Paulo R. DA C. POSSA[3]

Federal Institute of Education, Technology and Science of Rio Grande do Norte (1), Federal University of Rio Grande do Norte (2), University of Mons (3)

doi:10.15199/48.2017.01.63

Partial Reconfiguration Exploration Over P^{2}IP Architecture

Abstract. *P^2 IP is a real-time image and video processing architecture featuring reconfigurable runtime capabilities, low latency and high performance. However, low energy consumption and battery life are crucial when targeting portable devices. In some applications, not all processing elements are in use representing a power leak that a Partial Reconfiguration (PR) strategy could mitigate. To assess its impact, three image processing algorithms have been deployed in a variant of this architecture implemented in an FPGA. Measurements show that use of PR leads to energy savings of up to 45%.*

Streszczenie. *P^2 IP jest metoda przetwarzania obrazu i video w czasie rzeczywistym z dobrą jakością i małym opóźnieniem. W celu zmniejszenia poboru mocy opracowano strategię Partial Reconfiuguration PR oraz opracowano architekturę wykorzystując FPGA. Wykorzystanie strategii Partial Reconfiguration w architekturze P^{2}IP()*

Keywords: Low-power consumption, FPGA, Partial Reconfiguration, Embedded real-time video processing system
Słowa kluczowe: przetwarzanie obrazu, FPGA, Partial Reconfiguration

Introduction

The Programmable Pipeline Image Processor (P^{2}IP) is a systolic Coarse-Grained Reconfigurable Architecture (CGRA) for real-time video processing. It features low-latency systolic array inherent structures, runtime reconfigurable data-path, high-performance CG operators and short compilation times of software applications. Its data path, operating at the pixel clock frequency, can deliver, after the initial latency of a 3-line pipeline, one processed pixel per clock cycle [1, 2, 3]. The architecture processing core consists of identical Processing Elements (PEs). Each PE contains an optimized set of essential image processing operators (see Figure 1) that can be parameterized in run time by software, using virtual reconfiguration. The number and content of PEs is defined before synthesis. Thus, although available and contributing to the overall power consumption, not all PEs are in use depending on the processing performed on the video stream.

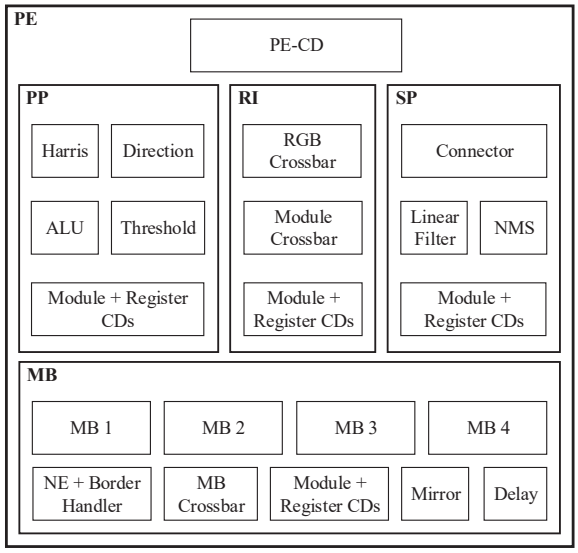

Fig. 1. Processing Element (PE) and its internal blocks. The main blocks are the Pixel Processor (PP), Memory Controller (MC), Spatial Processor (SP), Reconfigurable Internconnection (RI) and Configuration Decoder (PE-CD).

Applications for which there are power consumption restrictions, or where it is not possible to replace the battery that powers the circuit, such as a drone or a satellite, require circuit components with the highest energy-efficiency possible. Indeed, the use of smaller devices or a small number of enhanced devices reduces system cost and power consumption. On some modern FPGA devices, the Partial Reconfiguration (PR) is a feature that allows changing the configuration of part of the device while the rest continue to operate. This feature can improve logic density by removing functions that do not operate simultaneously in the FPGA. In the context of P^{2}IP, PR could lead to power savings by just replacing the content of an idle PE by a bypass using PR. This implies the use of a PE that implements no functionality other than a latched input driving its output. This way, if there is one or more idle PEs, these can, using PR, assume the bypass configuration, contributing to a lower power consumption.

We propose in this article a novel FPGA-based P^{2}IP implementation using PR to reduce energy consumption. During configuration, the content of a PE can now be replaced by a bypass core, plus the possibility of assigning an optimized functionality. To validate our proposal, we show comparative results concerning resource allocation, energy consumption and reconfiguration time for three reference applications.

Energy efficiency in FPGAs using PR

Many techniques can be used in order to achieve the same functionality with a more efficient power consumption in FPGAs.

[4] uses PR in a Zynq applied to video processing, and concludes that PR leads to a more efficient resource utilization and a lower power consumption.

[5] proves that one way to compensate the power consumption increase during PR is to maximize the partial bitstream transfer bandwidth from external memory to the PR interface.

[6] designs an intelligent ICAP controller, making use of DMA, to a Virtex-4 board. It is a good solution for Virtex-4, which does not support DMA when copying partial bitstreams, but imposes extra logic to synthesize the modified ICAP interface.

[7] implements a FIR filter applied to a Software-Defined Radio and concludes that using PR leads to a half of the original power consumption.

[8] shows graphics proving that a face recognition system using PR has a lower power consumption.

[9] proposes a 1-cycle reconfiguration scheme, although all reconfigurable elements are mapped into DSP48E1 ele-

ments. It is fast but associated to a high power consumption and a high end FPGA (and, consequently, expensive).

[10] describes an alternative way to load a partial bitstream in a Virtex-5 board. A customized PR controller is developed, which uses DMA to load the partial bitstream from external memory (DDR) to the ICAP interface, being more efficient than the traditional approach from Xilinx for the Virtex-5 family.

PR applied on P2IP

The runtime flexibility of P^{2}IP requires that the number of PEs and the provided functionality be enough to withstand all possible stream operations. For that reason, the number of PEs is defined before synthesis. Although, depending on the video processing algorithm to be executed, PEs are not completely in use. Considering, for instance, basic algorithms such as Edge Sharpening (*Sharp*), Canny Edge Detection (*Edge*) or Harris Corner Detection (*Corner*), they just share some operations. *Sharp* uses just three PEs to data processing, while the others use five (*Edge*) and seven (*Corner*), respectively. In that context, unused PEs still contribute to both, dynamic and static power consumption. For details about mapping each application onto P^{2}IP refer to [2].

To make possible the execution of the three aforementioned applications, seven PEs are defined. This number is chosen according to the Corner application, which, among the three, requires the greatest amount of PEs. At runtime, the PEs or its content cannot simply be removed: it would interrupt the video stream continuity. Thus, in addition to the regular content of a PE core (all the blocks as shown in Figure 1), a modified version, (core+bypass) is proposed, in which the output buffers the input, to ensure a continuous video stream, before removing its core. The core of each PE is contained in a Reconfigurable Region (RR), as suggested in [3]. So, seven RRs are defined in the FPGA area. PEs are equal in size and content, hence, all RRs resource requirements are the same. However, resources allocated to each RR may vary, depending on where the RR is allocated in the FPGA area (and, consequently, the available resources in the referred area).

To achieve the three mentioned applications using PR, three configurations are necessary:

- *Sharp*: RR1, RR2, RR3 in default configuration; RR4, RR5, RR6 and RR7 bypassed;
- *Edge*: RR1, ..., RR5 in default configuration; RR6 and RR7 bypassed;
- *Corner*: RR1, ..., RR7 in default configuration.

Figure 2 shows the seven PEs configured to execute the *Sharp* application. The software-driven configuration mechanism is responsible for activating the inputs, outputs and internal blocks of each PE.

Figure 3 shows *Canny* application mapping, where is possible to observe that the two last PEs are configured as bypass, contributing to a more efficient power consumption.

Figure 4 shows *Corner* application, in which all seven PEs are in the default configuration.

Results and discussion

Synthesis has been performed using Xilinx Vivado 2015.2.1 and Zynq 7020 System on Chip (SoC). It consists of a SoC containing an Artix-7 FPGA, from Xilinx, and a dual-core ARM Cortex-A9 processor. Figure 5 shows the floorplanning, containing the seven RRs and the interconnections between them. As can be seen in the aforementioned Figure, there are three RR sizes: the smallest (RR1, RR2 and RR3), the medium (RR4) and the biggest (RR5, RR6 and RR7). As

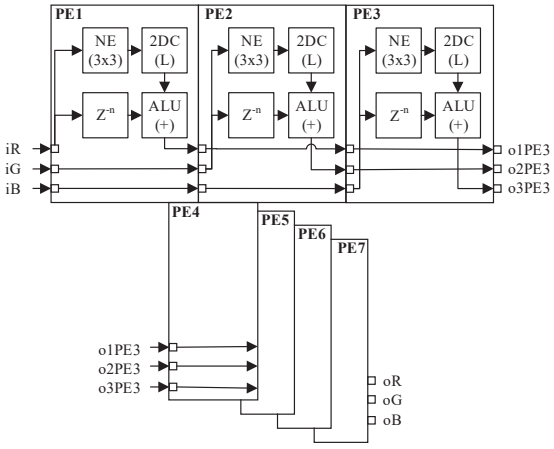

Fig. 2. *Sharp* application mapped onto P^{2}IP: the first three PEs are in default configuration; the four last are configured as bypass.

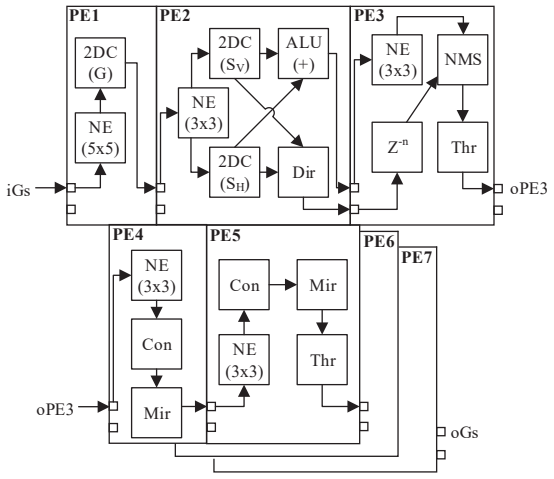

Fig. 3. *Canny* application mapped onto P^{2}IP: the first five PEs are in default configuration; the two last are configured as bypass.

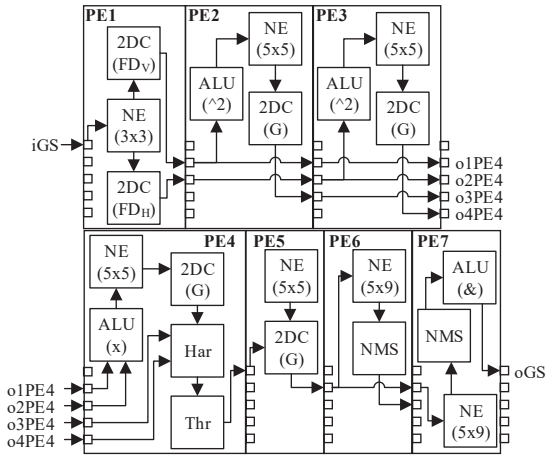

Fig. 4. *Corner* application: all PEs are in default configuration.

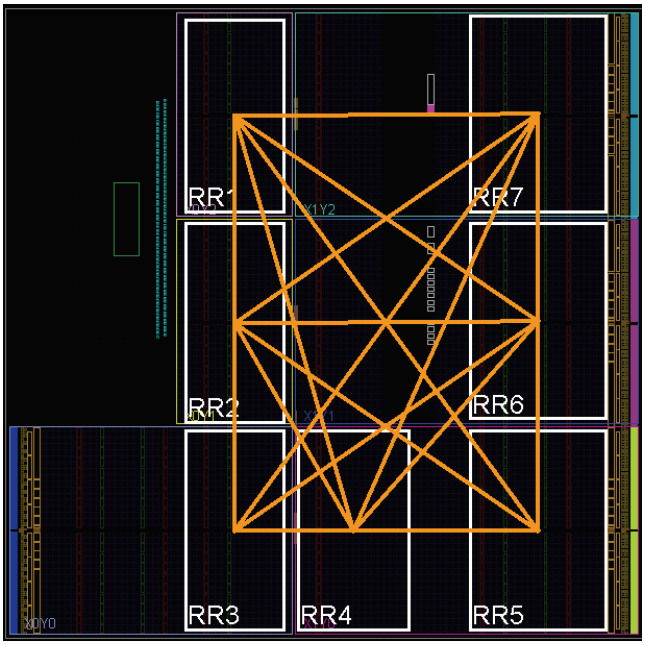

Fig. 5. Floorplanning, showing the seven RRs (white borders), and the corresponding connections among them (in orange).

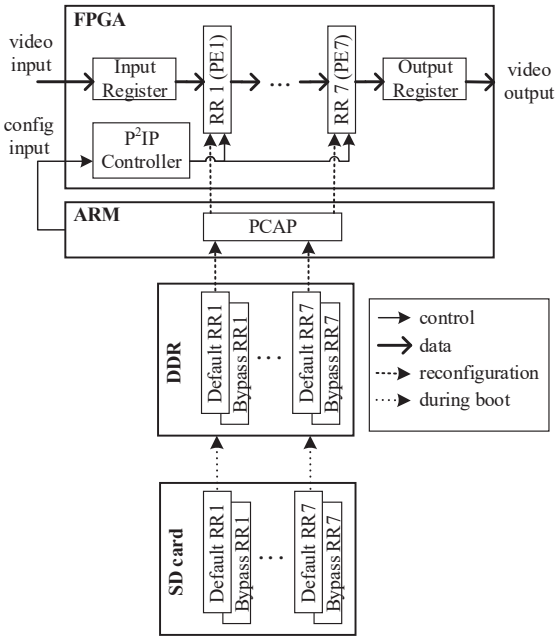

Fig. 6. P^{2}IP using PR, where the ARM loads the partial bitstreams from DDR and loads into the RRs.

explained before, resources necessary for each RR are the same, but resources availability throughout the FPGA area varies. So RR5, RR6 and RR7 need to be bigger due to resource limitation where these RRs were placed.

Initially, the fourteen partial bitstreams (Default RR1, ..., Bypass RR7, in Figure 6) are stored in a SD card. During boot, the ARM processor copies the fourteen partial bitstreams (Default RR1, ..., Bypass RR7) to the DDR memory. The ARM also loads one full bitstream in the FPGA. After loading the full bitstream the FPGA starts running.

To load a partial bitstream into a RR the first step is to copy the partial bitstream to the Processor Configuration Access Port (PCAP), from ARM, using DMA [11]. For more details about the bitstream copy from the SD card to the DDR memory and from the DDR memory to the PCAP interface (valid for the 7-series FPGAs from Xilinx) can be found on [12]. This process is more efficient than the one adopted by the previous FPGA families from Xilinx. Previous generations did not use DMA, turning the partial bitstream transfer slower [13]. PCAP interface is responsible for loading the partial bitstream into the RR.

The ARM is still used to activate the inputs, outputs and internal blocks of each PE, through an AXI4-Lite interface. For details about how the configuration mechanism works refer to [2]. Since all the video processing is done on the FPGA side, we have chosen to use a bare-metal implementation on the ARM side, instead of using an Operating System. Figure 6 shows the block diagram of the architecture using PR, detailing how the ARM loads a partial bitstream into P^{2}IP.

Among the three applications, *Sharp* requires less resources, since four out of seven PEs are partially reconfigured as bypass. *Edge* still uses less logic resources than the static implementation, since the two last PEs are bypassed. *Corner* is more resource-consuming than the static implementation because all PEs are in the default configuration.

This application demands a higher number of resources than the original implementation, but in the worst case the resource increase is less than 5%, due to the extra logic added when using PR, and, in the best case, there is a resource utilization reduction of more than 50%.

Table 1 shows the resources utilized by each application compared to the static implementation. *Sharp* requires less than a half resources, when compared to the original implementation. *Edge* uses less than 80% of the resources required by the static implementation. *Corner* introduces almost 5% more FFs and 2% LUTs than the static implementation.

Table 1. Allocated resources, compared to the original implementation.

	LUTs	FFs	RAMB18
Sharp	43.59%	48.31%	42.85%
Edge	70%	76.62%	71.42%
Corner	101.94%	104.73%	100%

Previously, it has been shown that *Sharp* and *Edge* use less active resources than the static implementation. It leads to energy savings. To measure the energy consumption, we used the ZC702 board from Xilinx, which has current and voltage monitoring circuits [14]. One of these circuits is able to measure current and voltage applied to the FPGA core. These monitoring circuits are connected to an I^{2}C bus, and can be accessed by the ARM [15] or by means of a USB Interface Adapter, from Texas Instruments [16]. In this work we have used the USB Interface Adapter. Fusion Power Digital Design software, from Texas Instruments, links to the USB Interface Adapter and gets voltage and current information, making possible to calculate the power consumption. An advantage of this method compared to the ARM reading current and voltage is that the first does not interfere in the ARM power consumption [17].

Table 2 shows the measured power for the three configurations using PR (third column), compared to the original implementation (second column). The last column of the referred Table shows how much power savings it is possible to achieve using PR into P^{2}IP. For each configuration 200 samples have been acquired using a sample rate of 100ms, totalizing a 20s acquisition (for each application). The values shown on Table 2 are the average of the 200 samples.

As can be seen in the previous Table, the power over-

Table 2. Measured power.

	Original	PR	Diff
Sharp	371mW	204mW	-45.01%
Edge	371mW	280mW	-24.52%
Corner	371mW	373mW	+0.54%

head for the *Corner* algorithm, due to the extra partial reconfiguration logic, is negiglible.

To demonstrate the efficiency in applying PR to P^{2}IP, we consider the specification of a commercial smartphone battery: 3.8V, 2.6Ah.

(1)
$$E_b = P_b \times t = 3.8V \times 2.6A \times 1h = 9.88W \times 3600s = 35.6kJ$$

From this result it is possible to calculate the autonomy when this battery is powering the system in configurations original, *Sharp*, *Edge* and *Corner*, as can be seen in Table 3, where it is possible to observe almost 82% autonomy increase (*Sharp* using PR) compared to the original implementation.

Table 3. Battery autonomy in original and PR implementations.

	Original	PR	Gain
Sharp		48.47h	+81.88%
Edge	26.65h	35.32h	+32.53%
Corner		26.51h	-0.015%

Another important point to be discussed is the amount of time necessary for changing configurations. Transitions require loading two partial bitstreams (such as from *Sharp* to *Edge*) or four partial bitstreams (such as from *Sharp* to *Corner*). According to Xilinx the bitstream transfer rate using PCAP interface in non-secure mode is 400 MB/s [18]. Partial bitstream size for RR1, RR2 and RR3 is 306KB; for RR4, 309KB; and, for RR5, RR6 and RR7, 409KB. To measure the time necessary to load one partial bitstream a 64-bit general purpose ARM timer was used. Time to reconfigure each partial bitstream was measured and the average data rate is 128.51 MB/s.

Table 4 shows the time necessary to change configurations. To load one partial bitstream it is necessary: 2.381ms, for RR1, RR2 or RR3; 2.404ms, for RR4; and 3.175ms for RR5, RR6 and RR7.

Table 4. Latency, when changing configurations.

	Sharp	*Edge*	*Corner*
Sharp	-	5.579ms	11.929ms
Edge	5.579ms	-	6.350ms
Corner	11.929ms	6.350ms	-

To prove that the latency when reconfiguring does not compromise the functioning of the system an analysis has been carried out based on the FullHD 1080p60 resolution, which has a frame period of 16.667ms. The slowest transition is from Sharp to Edge, which takes 11.929ms.

Equation 2 relates total latency (t_{total}), where t_RP is the PR latency (see Table 4), t_{config} is the configuration mechanism latency and N_{config} is the number of operators to be configured, considering that after applying PR one configuration message (for each operator) is sent to the respective PE to active the inputs, outputs and internal blocks.

(2)
$$t_{total} = t_{RP} + t_{config} + N_{config}$$

According to [3] $t_{config} = 0,34\mu s$ for each operator to be configured. In terms of latency the worst case is to change from Sharp to Corner, in which it is necessary to apply PR to four RRs and configure 21 operators (see Figure 4). In this case $t_{total} = 11.937ms$. So, only one frame will be lost when applying PR. When changing the configuration (the number of active PEs or event an internal block) of the architecture (using PR or not) one frame will be lost. This work proves that it is possible to apply PR without loosing additional frames.

Conclusion

In this article we have presented a low-power version of P^{2}IP architecture based on PR usage. Three image processing algorithms were mapped on the proposed architecture using PR in order to validate it. Experiments shown that additional resources required by PR process are irrelevant. Power consumption comparison of original and PR implementations has been carried out and attested that PR implementation leads to power savings.

Future work should investigate creating not only two configurations for each PE (default and bypass), but one configuration for each PE variation, to check how much more pwoer savings it is possible to achieve. This way each PE variation would contain only the internal blocks used for a specific processing.

***Authors**: Prof. MSc. Alvaro M. AVELINO, Federal Institute of Education, Science and Technology of Rio Grande do Norte, RN120 - Alto de Santa Luzia, Nova Cruz/RN, Brazil, ZIP 59215-000, e-mail alvaro.medeiros@ifrn.edu.br. Prof. PhD. Valentin O. RODA, Department of Electrical Engineering, Federal University of Rio Grande do Norte, Natal/RN, Brazil, ZIP 59072-970, e-mail valentin@ct.ufrn.br. Prof. PhD. Carlos A. V. SAKUYAMA, MSc. Glauberto A. L DE ALBUQUERQUE, PhD. Paulo R. DA C. POSSA, Department of Electronics and Microelectronics, University of Mons, Mons, Belgium, ZIP 7000, e-mail carlos.valderrama@umons.ac.be, glaulberto.albuquerque@umons.ac.be, ppossa@scarlet.be.*

REFERENCES

[1] P. Possa, S. Mahmoudi, N. Harb, C. Valderrama and P. Manneback, “A multi-resolution FPGA-based architecture for real-time edge and corner detection”, IEEE Transactions on Computers, 63(10), pp. 2376-2388, 2014.
[2] P. Possa, N. Harb, E. Dokladalova and C. Sakuyama, “P2IP: A novel low-latency Programmable Pipeline Image Processor”, Microprocessors and Microsystems, 39(7), pp. 529-540, 2015.
[3] P. Possa, “Reconfigurable low-latency architecture for real-time image and video processing”, UMONS, Jun 2013.
[4] N. Nayak, “Accelerated Computation using Runtime Partial Reconfiguration”, University of Stuttgart, Nov 2013.
[5] S. Liu, R. Pittman and A. Forin, “Energy Reduction with Run-Time Partial Reconfiguration”, Microsoft Research, 2009.
[6] S. Liu, R. Pittman and A. Forin, “Minimizing Partial Reconfiguration Overhead with Fully Streaming DMA Engines and Intelligent ICAP Controller”, Microsoft Research, 2009.
[7] T. Becker, W. Luk and P. Cheung, “Energy-aware optimization for run-time reconfiguration, 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)”, pp. 55-62, 2010.
[8] A. Ahmad, A. Amira, P. Nicholl and B. Krill, “Dynamic Partial Reconfiguration of 2-D Haar Wavelet Transform (HWT) for face recognition systems, IEEE 15th International Symposium on Consumer Electronics”, pp. 9-13, 2011.
[9] A. Ihsen, “Design of Self-Tuning Reliable Embedded Systems and its Application in Railway Transportation Systems”, Université de Valenciennes, Apr 2016.
[10] J. Dondo, J. Barba, F. Rincón, F. Moya and J. López, “Dynamic objects: Supporting fast and easy run-time reconfiguration in FPGAs”, Journal of Systems Architecture, 65(5), pp. 1484-1493, 2016.
[11] Xilinx, “Vivado Design Suite Tutorial - Partial Reconfiguration”, UG947, 2015.

[12] M. Kadi, P. Rudolph, D. Gohringer and M. Hubner, “Dynamic and partial reconfiguration of Zynq 7000 under Linux”, 2013 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2013, pp. 1-5, 2013.
[13] B. Blodget, C. Bobda, M. Hubner and A. Niyonkuru, “Partial and Dynamically Reconfiguration of Xilinx Virtex-II FPGAs”, Field Programmable Logic and Application: 14th International Conference, FPL 2004 Proceedings, pp. 801-810, 2004.
[14] Xilinx, “ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC User Guide”, UG850, 2015.
[15] E. Srikanth, “Zynq-7000 AP SoC Low Power Techniques part 3 - Measuring ZC702 Power with a Standalone Application Tech Tip”, 2014.
[16] Texas Instruments, “USB Interface Adapter Evaluation Module User’s Guide”, 2006.
[17] J. Nunez-Yanez, M. Hosseinabady and A. Beldachi, “Energy Optimization in Commercial FPGAs with Voltage, Frequency and Logic Scaling”, IEEE Transactions on Computers, 59(1), pp. 1-15, 2013.
[18] Xilinx, “Partial Reconfiguration of a Hardware Accelerator on Zynq-7000 All Programmable SoC Devices”, 2013.