

# LP-P<sup>2</sup>IP: A Low-Power Version of P<sup>2</sup>IP Architecture Using Partial Reconfiguration

Álvaro Avelino<sup>1</sup>(✉), Valentin Obac<sup>2</sup>, Naim Harb<sup>3</sup>, Carlos Valderrama<sup>3</sup>,  
Glauberito Albuquerque<sup>3</sup>, and Paulo Possa<sup>3</sup>

<sup>1</sup> Technology and Science of Rio Grande do Norte,  
Federal Institute of Education, Nova Cruz, RN, Brazil  
`alvaro.medeiros@ifrn.edu.br`

<sup>2</sup> Electrical Engineering Department,  
Federal University of Rio Grande do Norte, Natal, RN, Brazil

<sup>3</sup> Electronics and Microelectronics Department, Université de Mons, Mons, Belgium

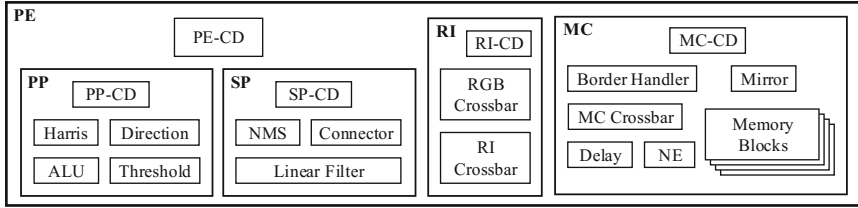
**Abstract.** Power consumption reduction is crucial for portable equipments and for those in remote locations, whose battery replacement is impracticable. P<sup>2</sup>IP is an architecture targeting real-time embedded image and video processing, which combines runtime reconfigurable processing, low-latency and high performance. Being a configurable architecture allows the combination of powerful video processing operators (Processing Elements or PEs) to build the target application. However, many applications do not require all PEs available. Remaining idle, these PEs still represent a power consumption problem that Partial Reconfiguration can mitigate. To assess the impact on energy consumption, another P<sup>2</sup>IP implementation based on Partial Reconfiguration was developed and tested with three different image processing applications. Measurements have been made to analyze energy consumption when executing each of three applications. Results show that compared to the original implementation of the architecture use of Partial Reconfiguration leads to power savings of up to 45%.

**Keywords:** Energy efficiency · Low-power consumption · FPGA · Partial reconfiguration · Embedded real-time video processing system

## 1 Introduction

The Programmable Pipeline Image Processor (P<sup>2</sup>IP) is a systolic Coarse-Grained Reconfigurable Architecture (CG-RA) for real-time video processing embedded in FPGA. 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 [2–4]. The architecture processing core consists of identical

Processing Elements (PEs). Each PE contains an optimized set of essential image processing operators (see Fig. 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 Interconnection (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, Partial Reconfiguration (PR) is a feature that allows changing the configuration of part of the device while the rest continues to operate. This feature can improve logic density by removing functions that do not operate simultaneously in the FPGA. In the context of P<sup>2</sup>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. In a certain application, if there is one or more idle PEs, these can, using PR, assume the bypass configuration, reducing the overall power consumption.

We propose in this article a novel FPGA-based P<sup>2</sup>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. The latter represents a future enhancement as heterogeneous PEs can extend the architecture to support novel processing capabilities. To validate our proposal, we show comparative results concerning resources allocation, energy consumption and reconfiguration latency for three reference applications.

## 2 Energy Efficiency in FPGAs

Power consumption is a combination of static (which depends on the temperature  $T$ ) and dynamic power, as stated by (1). The static power is caused by leakage

currents inside transistors and the dynamic power is caused by the switching activity when charging and discharging the load capacitance  $C$ , as well as short-circuit currents when transistors commute.

$$P_{total} = P_{static}(T) + P_{dynamic}(f) \quad (1)$$

Dynamic power, as stated in (2), has linear dependency on the clock frequency  $f$  and a quadratic dependency on the supply voltage  $V$ . In an FPGA, the load capacitance depends on the number of logic and routing elements used. The factor  $\alpha$  is the activity or toggle rate of an element; it depends on the topology and its input stimuli.

$$P_{dynamic} = \alpha \times C \times V^2 \times f \quad (2)$$

## 2.1 Related Works

The Partial Reconfiguration capability can be beneficial to P<sup>2</sup>IP in two aspects, not only it can help to reduce power consumption but also extend its original functionality. Indeed, we report many related techniques that can be used in FPGAs to achieve more efficient power consumption while preserving functionality. However, while PR allows the reuse of the underlying logic, design granularity, reconfiguration support infrastructure and reconfiguration speed may be limiting factors.

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]. In [5], the authors propose an intelligent Internal Configuration Access Port (ICAP) controller using DMA for a Virtex-4 board. This is a good solution for Virtex-4, which does not support DMA when copying partial bitstreams, but imposes additional logic to synthesize the modified ICAP interface. [18] 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.

Concerning granularity and reconfiguration speed, in [7] it is proposed a 1-cycle reconfiguration scheme, although all reconfigurable elements are mapped into DSP48E1 cells. Thus, a fast reconfiguration can be carried out by updating the parameters of the DSP cells, but at the cost of high power consumption and high-end (and therefore costly) FPGA. Another approach, [1], proposes an alternative ICAP interface (called AC-ICAP) capable of applying PR to single LUTs without requiring pre-computed partial bitstreams. According to the authors, it imposes an acceleration of 380x with respect to the Xilinx ICAP controller. The disadvantage is that it consumes 5% additional cells on a Virtex-5 FPGA. With regard to structures using more complex PR components, the authors in [8] implement a FIR filter applied to Software-Defined Radio and conclude that using PR leads to a half of the original power consumption.

Compared to the works mentioned above P<sup>2</sup>IP is already a software-configurable and customizable hardware architecture. This implies that it is

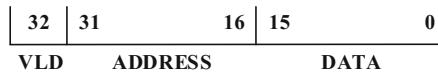
already inherently scalable and flexible, so we assume a very small resources overhead by supporting PR. Due to requiring few or no additional controllers, when extended, its impact on energy consumption will be limited. As demonstrated by previous works, granularity has an impact on the size of partial bit-streams as well as on reconfiguration speed. The proposed architecture using regular components of intermediate granularity (greater than a DSP cell or a LUT) maintains PR time and partial bit-streams restricted. Moreover, the scalable aspect of the architecture makes it possible to combine various strategies to save static and dynamic power. However, this is a real-time configurable architecture, so, PR time can have an impact on the resulting image stream processing. Since the reconfiguration task is executed by the software side and several partial bitstream loading strategies are already available, the potential need to speed up the PR process is left out of the scope of this work.

### 3 Modifications on P<sup>2</sup>IP

The original P<sup>2</sup>IP architecture was enhanced to be used as AXI compliant IP for an FPGA implementation with extended configurable functionality including PR (for details about the original implementation refer to [4]).

#### 3.1 Configuration Mechanism

The configuration mechanism allows to enable/disable operators as well as the input/outputs of a PE. It consists of a configuration tree composed by Configuration Decoders (CDs) organized hierarchically. In the original version it communicates via an 8-bit serial interface [3]. The extended version provides an AXI4-Lite 32-bit interface clocked at 100 MHz and the configuration word carries two bytes of data (instead of one in the original implementation), as shown in Fig. 2.



**Fig. 2.** Configuration word. The Configuration Block reads the word when the VLD bit is high. ADDRESS corresponds to the operator ID, and DATA is the configuration info.

On Fig. 1 it is possible to see the PE Configuration Decoder (PE-CD) and four (Module + Register) dedicated CDs (for the PP, RI, SP and the MC modules). Each CD can decode one (e.g. NE), two (e.g. ALU) or three words of data (e.g. Reconfigurable Interconnect crossbar), and this limit is defined during instantiation of the component.

### 3.2 PR Applied on P<sup>2</sup>IP

The runtime flexibility of P<sup>2</sup>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<sup>2</sup>IP refer to [3].

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 [3]. 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 Fig. 1), a modified version (core+bypass) is proposed, in which the output buffers the input, to ensure a continuous video stream before removing the core. Indeed, the core of each PE is contained in a Reconfigurable Region (RR), as suggested in [4]. 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 examples using PR, three configurations are defined:

- *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.

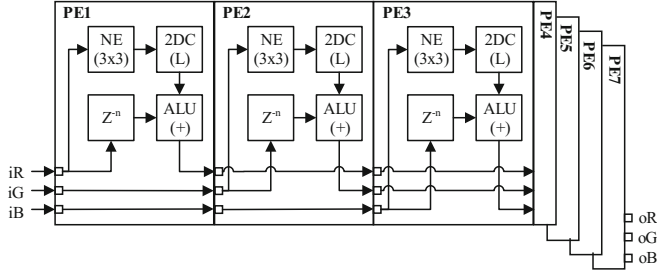
Figures 3, 4 and 5 show, respectively *Sharp*, *Canny* and *Corner* applications mapped onto P<sup>2</sup>IP using PR. The software-driven configuration mechanism is responsible for activating the inputs, outputs and internal blocks of each PE.

## 4 Methodology

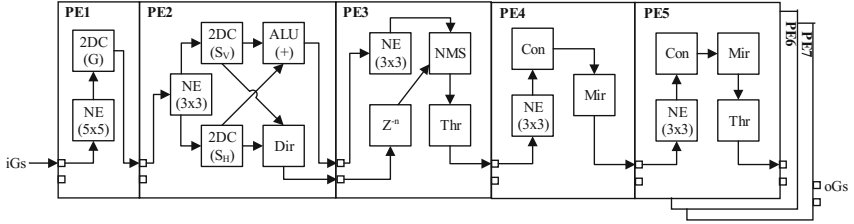
The new architecture is able to allocate resources (PEs) to reconfigurable regions (RRs) defined in the FPGA area. Resources allocated to each RR can be of type bypass or original PE core.

Fourteen partial bitstreams (Default RR1..7 and Bypass RR1..7, in Fig. 6) are initially stored in an SD card. During boot, the ARM processor copies these partial bitstreams to the DDR memory.

After that, the ARM also loads a full bitstream (the initial configuration containing static and dynamic parts) before the FPGA starts running.

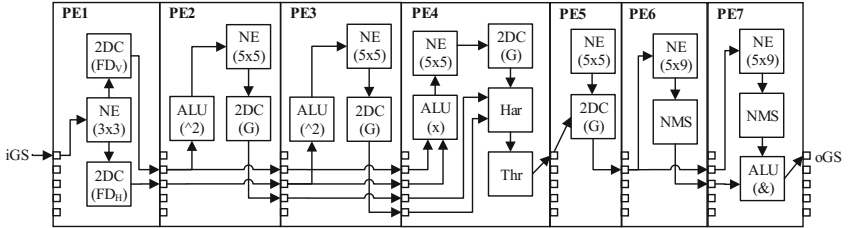


**Fig. 3.** *Sharp* application mapped onto P<sup>2</sup>IP: the first three PEs are in default configuration; the four last are configured as bypass.

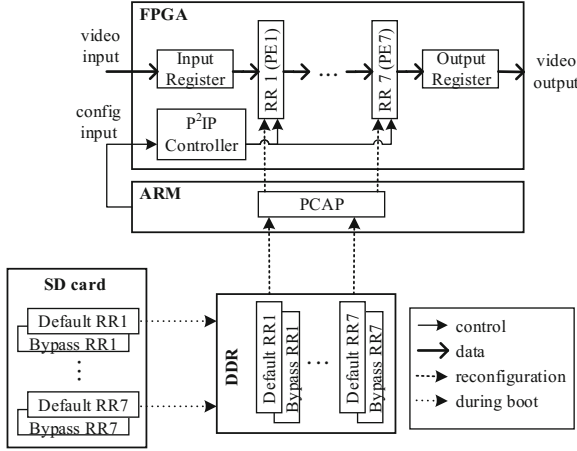


**Fig. 4.** *Canny* application mapped onto P<sup>2</sup>IP: the first five PEs are in default configuration; the two last are configured as bypass.

By default, the Xilinx Zynq platform offers two options to load a bitstream into the FPGA: the Internal Configuration Access Port (ICAP) or the Processor Configuration Access Port (PCAP). The first one is in use, for a long time, by the previous FPGA families [1, 5, 17, 18]. It consists of an IP softcore and, consequently, spends some FPGA resources. The PCAP interface is native, does not consume any FPGA resources and uses a DMA engine [10]. This process is more efficient than the one adopted by the previous Xilinx FPGA families, since these generations did not use DMA natively, turning the partial bitstream transfer slower [9] while forcing the designer to consume more FPGA resources to allocate a custom DMA engine or the ICAP interface [17].



**Fig. 5.** *Corner* application: all PEs are in default configuration.



**Fig. 6.** P<sup>2</sup>IP using PR: during boot, the ARM reads the partial bitstreams from the SD Card and loads them into the DDR. On demand, during runtime, the partial bitstreams are loaded from DDR into the RRs.

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 Xilinx 7-series FPGAs) can be found on [19].

Since the purpose of this work is to reduce energy consumption, additional logic must be minimized, therefore the PCAP interface was chosen to transfer (static and partial) bitstreams from the memory to the FPGA, under the ARM supervision. The ARM is also used to activate the inputs/outputs, internal inter-connections and blocks of each PE via an AXI4-Lite [14] interface.

For details about how the configuration mechanism works refer to [3]. 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<sup>2</sup>IP.

## 5 Results

A P<sup>2</sup>IP implementation based on PR was developed and tested with multiple configuration scenarios. Synthesis has been performed using 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. To demonstrate that the use of PR applied to P<sup>2</sup>IP implies energy savings without disrupting the real-time feature of the architecture, results regarding resource allocation, power measurement and reconfiguration time analysis are shown in this section.

## 5.1 Resource Analysis

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%.

The left side of 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 (left side) and measured power, in mW (right side).

	Allocated resources			Measured power, in mW		
	LUTs	FFs	RAMB18	Original	PR	$\Delta$
<i>Sharp</i>	43.59%	48.31%	42.85%	371	204	−45.01%
<i>Edge</i>	70%	76.62%	71.42%		280	−24.52%
<i>Corner</i>	101.94%	104.73%	100%		373	+0.54%

## 5.2 Power Consumption Measurement

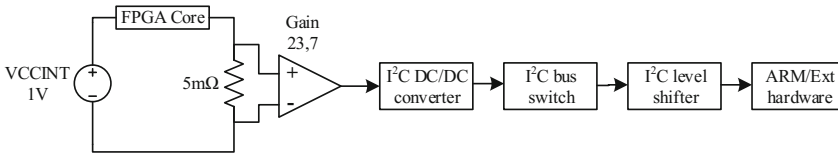
Previously, it has been shown that *Sharp* and *Edge* use less active resources than the static implementation. It leads to energy savings. To measure the energy consumption, we used the ZC702 board from Xilinx, which has current and voltage monitoring circuits [11]. One of these circuits is able to measure current and voltage applied to the FPGA core, as shown in Fig. 7. *VCCINT* is a 1 V voltage applied to the FPGA core. The voltage drop across a 5 m $\Omega$  is fed to an Instrumentation Amplifier (IA), whose gain is 23.7. The IA output serves as input to an I<sup>2</sup>C DC/DC converter, which monitors *VCCINT* and turns the analog voltage into digital data (I<sup>2</sup>C). Since there are other I<sup>2</sup>C components on the board, an 1-to-8 channel I<sup>2</sup>C multiplexer is present.

Data can be accessed by the ARM [12], the FPGA or by means of a USB Interface Adapter, from Texas Instruments [13]. As stated before the monitored information is accessed through I<sup>2</sup>C protocol. Getting data using the FPGA is not the most efficient solution, since it means adding logic resources to define in hardware I<sup>2</sup>C communication interface and, consequently, it would contribute to



power consumption increasing. Using the ARM is a good alternative if the USB Interface Adapter is not available.

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. It is possible to define measurement parameters and acquisition rate. Minimum acquisition rate is 10 ms, but it is important to highlight that the USB Interface Adapter is plugged to a computer running Microsoft Windows, which is not a Real Time Operating System (RTOS), and, thus, there is no guarantee that the acquisition rate will be respected. Due to this restriction during tests the minimum acquisition rate used was 100 ms. 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 [15].



**Fig. 7.** FPGA core current measurement circuit on ZC702 board. Current can be read by the ARM processor or by an external hardware from Texas Instruments, both through the I<sup>2</sup>C bus.

The right side of Table 1 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<sup>2</sup>IP. For each configuration 200 samples have been acquired using a sample rate of 100 ms, totalizing a 20s acquisition (for each application). The values shown on Table 1 are the average of the 200 samples.

As can be seen in the previous Table, the power overhead for the *Corner* algorithm, due to the extra partial reconfiguration logic, is negligible.

### 5.3 Reconfiguration Latency

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 [16]. Partial bitstream size for RR1, RR2 and RR3 is 306 KB; for RR4, 309 KB; and, for RR5, RR6 and RR7, 409 KB. 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 2 shows the time necessary to change configurations. To load one partial bitstream it is necessary: 2.381 ms, for RR1, RR2 or RR3; 2.404 ms, for RR4; and 3.175 ms for RR5, RR6 and RR7.

**Table 2.** Latency, when changing configurations, in ms.

	<i>Sharp</i>	<i>Edge</i>	<i>Corner</i>
<i>Sharp</i>	-	5.579	11.929
<i>Edge</i>	5.579	-	6.350
<i>Corner</i>	11.929	6.350	-

To assure that, using PR, the system remains a real time one, the following Equation is used:

$$t_{total} = t_{reconfig} + t_{config} \quad (3)$$

where  $t_{total}$  is the total reconfiguring latency,  $t_{reconfig}$  is the time necessary to apply PR to the RRs and  $t_{config}$  is the time necessary to configuring internal PE blocks.

Time necessary to apply PR depends on how many RRs will be configured and is described in (4):

$$t_{reconfig} = \sum t_{RR_i} \quad (4)$$

where  $t_{RR_i}$  is the time necessary to apply PR to each RR.

Time necessary to apply PR to one RR depends on the external memory (which stores the partial bitstream) access and also on the time to load the partial bitstream on the respective RR and is shown in (5):

$$t_{RR_i} = t_{DDR} + t_{load_{PB}} \quad (5)$$

To maintain the real time feature of the system the following Equation must be respected:

$$t_{total} < t_{frame} \quad (6)$$

If (6) is respected then only one frame will be lost during reconfiguration, using PR or not. If the time overhead introduced by PR is less than the frame timing the extra time necessary to apply PR is admissible and does not imply in additional delay, that is, the system remains real time.

It is necessary  $t_{config} = 0.27 \mu s$  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 Fig. 5). In this case  $t_{total} = 11.935 ms$ . 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 losing additional frames.

## 6 Conclusions

In this article we have presented a low-power P<sup>2</sup>IP architecture based on the use of the PR strategy. The original architecture was extended to support PR: processing elements were designated as PR components resulting on less than 5% resources overhead. To demonstrate the advantages of this novel architecture in terms of power consumption, three image processing algorithms were mapped and executed on both architectures. Power consumption comparison of original and PR implementations has been carried out and attested that PR implementation leads to power savings of up to 45%. The worst-case scenario, which takes into account the use of all available resources, implies an additional energy cost of less than 1%. Furthermore, PR latency does not affect the real-time feature of the system.

The PR strategy should not only be applied to lower the power consumption, it also serves to combine multiple alternative implementations of PEs that can be interchanged according to particular execution and quality requirements. Thus, future work should investigate the balance between power saving and required processing power.

**Acknowledgments.** The authors would like to thank the support from the Coordination of Superior Level Staff Improvement (CAPES), Brazilian sponsoring agency, and also the Electronics and Microelectronics Department from the University of Mons, Belgium, for the support offered to the development of this work.

## References

1. Cardona, L.A., Ferrer, C.: AC-ICAP: a flexible high speed ICAP controller. *Int. J. Reconfigurable Comput.* (2015). doi:[10.1155/2015/314358](https://doi.org/10.1155/2015/314358)
2. Possa, P.R., Mahmoudi, S.A., Harb, N., Valderrama, C., Manneback, P.: A multi-resolution FPGA-based architecture for real-time edge and corner detection. *IEEE Trans. Comput.* **63**, 2376–2388 (2014). doi:[10.1109/TC.2013.130](https://doi.org/10.1109/TC.2013.130)
3. Possa, P., Harb, N., Dokládlová, E., Valderrama, C.: P<sup>2</sup>IP: a novel low-latency programmable pipeline image processor. *Microprocess. Microsyst.* **39**, 529–540 (2015). doi:[10.1016/j.micpro.2015.06.010](https://doi.org/10.1016/j.micpro.2015.06.010)
4. da Cunha Possa, P.R.: Reconfigurable low-latency architecture for real-time image and video processing. UMONS (2013)
5. Liu, S., Pittman, R.N., Forin, A.: Minimizing partial reconfiguration overhead with fully streaming DMA engines and intelligent ICAP controller. In: Microsoft Research (2009)
6. Liu, S., Pittman, R.N., Forin, A.: Energy reduction with run-time partial reconfiguration. In: Microsoft Research (2009)
7. Ihnen, A.: Conception de Systèmes Embarqués Fiabiles et Auto-réglables. Université de Valenciennes, Applications sur les Systèmes de Transport Ferroviaire (2016)
8. Becker, T., Luk, W., Cheung, P.Y.K.: Energy-aware optimization for run-time reconfiguration. In: 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 55–62 (2010). doi:[10.1109/FCCM.2010.17](https://doi.org/10.1109/FCCM.2010.17)

9. Blodget, B., Bobda, C., Huebner, M., Niyonkuru, A.: Partial and dynamically reconfiguration of Xilinx Virtex-II FPGAs. In: Becker, J., Platzner, M., Vernalde, S. (eds.) FPL 2004. LNCS, vol. 3203, pp. 801–810. Springer, Heidelberg (2004). doi:[10.1007/978-3-540-30117-2\\_81](https://doi.org/10.1007/978-3-540-30117-2_81)
10. Xilinx: Vivado Design Suite Tutorial - Partial Reconfiguration (2015)
11. Xilinx: ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC User Guide (2015)
12. Srikanth, E.: Zynq-7000 AP SoC Low Power Techniques part 3 - Measuring ZC702 Power with a Standalone Application Tech Tip (2014)
13. Texas Instruments: USB Interface Adapter Evaluation Module User's Guide (2006)
14. Xilinx: AXI4-Lite IPIF v3.0 (2016)
15. Nunez-Yanez, J.L., Hosseinabady, M., Beldachi, A.: Energy optimization in commercial FPGAs with voltage, frequency and logic scaling. *IEEE Trans. Comput.* **65**, 1484–1493 (2016). doi:[10.1109/TC.2015.2435771](https://doi.org/10.1109/TC.2015.2435771)
16. Xilinx: Partial Reconfiguration of a Hardware Accelerator on Zynq-7000 All Programmable SoC Devices (2013)
17. Silva, C.A.A., Neto, A.D.D., Oliveira, J.A.N., Melo, J.D., Barbalho, D.S., Avelino, A.M.: Definition of an architecture to configure artificial neural networks topologies using partial reconfiguration in FPGA. *IEEE Latin Am. Trans.* **15**, 2094–2100 (2015)
18. Dondo, J.D., Barba, J., Rincón, F., Moya, F., López, J.C.: Dynamic objects: supporting fast and easy run-time reconfiguration in FPGAs. *J. Syst. Archit.* **59**, 1–15 (2013). doi:[10.1016/j.sysarc.2012.09.001](https://doi.org/10.1016/j.sysarc.2012.09.001)
19. Muhammed, A.K., Rudolph, P., Gohringer, D., Hubner, M.: Dynamic and partial reconfiguration of Zynq 7000 under Linux. In: 2013 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2013 (2013). doi: [10.1109/ReConFig.2013.6732279](https://doi.org/10.1109/ReConFig.2013.6732279)

Applied Reconfigurable Computing

13th International Symposium, ARC 2017, Delft, The Netherlands, April 3-7, 2017, Proceedings

Wong, S.; Beck, A.C.; Bertels, K.; Carro, L. (Eds.)

2017, XX, 332 p. 142 illus., Softcover

ISBN: 978-3-319-56257-5