energies

Article

Energy Efficiency Evaluation of Dynamic Partial Reconfiguration in Field Programmable Gate Arrays: An Experimental Case Study

Vincenzo Conti [1,*], Leonardo Rundo [2,3], Giuseppe Dario Billeci [1], Carmelo Militello [3] and Salvatore Vitabile [4]

1 Faculty of Engineering and Architecture, University of Enna KORE, 94100 Enna, Italy; giuseppedario.billeci@unikorestudent.it
2 Department of Informatics, Systems and Communication (DISCo), University of Milano-Bicocca, 20126 Milan, Italy; leonardo.rundo@disco.unimib.it
3 Institute of Molecular Bioimaging and Physiology (IBFM), Italian National Research Council (CNR), 90015 Cefalù, Italy; carmelo.militello@ibfm.cnr.it
4 Department of Biopathology and Medical Biotechnologies (DIBIMED), University of Palermo, 90133 Palermo, Italy; salvatore.vitabile@unipa.it
* Correspondence: vincenzo.conti@unikore.it; Tel.: +39-0935-536445

Received: 25 January 2018; Accepted: 22 March 2018; Published: 24 March 2018

Abstract: Both computational performances and energy efficiency are required for the development of any mobile or embedded information processing system. The Internet of Things (IoT) is the latest evolution of these systems, paving the way for advancements in ubiquitous computing. In a context in which a large amount of data is often analyzed and processed, it is mandatory to adapt node logic and processing capabilities with respect to the available energy resources. This paper investigates under which conditions a partially reconfigurable hardware accelerator can provide energy saving in complex processing tasks. The paper also presents a useful analysis of how the dynamic partial reconfiguration technique can be used to enable energy efficiency in a generic IoT node that exploits a Field Programmable Gate Array (FPGA) device. Furthermore, this work introduces a hardware infrastructure and new energy metrics tailored for the energy efficiency evaluation of the dynamic partial reconfiguration process in embedded FPGA based devices. Exploiting the ability of reconfiguring circuit portions at runtime, the latest generation of FPGAs can be used to foster a better balance between energy consumption and performance. More specifically, the design methodology for the implemented digital signal processing application was adapted for the ZedBoard. To this aim, a case study of a video filtering system is proposed and analyzed by dynamically loading three different hardware filters from the management software running on a Linux-based device. With more details, the presented analytical framework allows for a direct comparison between the energy efficiency of a dynamic partially reconfigurable device and a static non-reconfigurable one. The estimated timing conditions that allow the dynamic partially reconfigurable process to achieve relevant energy efficiency with respect to the corresponding static architecture are also outlined.

Keywords: energy efficiency; dynamic partial reconfiguration; Field Programmable Gate Array; digital signal processing; video filtering

1. Introduction

The diffusion of electronic systems in any kind of environment has laid the foundations for ubiquitous computing. To push the potential of this paradigm to the limit, it was necessary to connect the objects together. For this purpose, the Internet has always provided a standard platform for world connectivity: from the first mobile networks, to the World Wide Web, to the Internet of Things (IoT). IoT enables connections among either objects and objects, or users and objects [1,2].

In a context where the number of complex embedded systems is ever increasing, it is necessary to have cutting-edge and powerful technologies that can support this paradigm. Such an example is the development of interfaces combined with objects, such as Radio Frequency Identification (RFID) and Wireless Sensor Networks (WSNs). These technologies represent a valuable example of how information and communication systems have been incorporated, even transparently, into the IoT objects around us. However, these sensing nodes are typically characterized by limited power resources. Therefore, digital signal acquisition and processing must be performed with high efficiency and low power consumption [3].

Current platforms are the result of generations of systems that have enabled the manipulation of big data: managing authentication, storage, processing, presentation of such information in a clear, efficient, and easy way [4,5]. In this context, the IoT paradigm may emerge successfully, since there is a need to overcome the traditional mobile computing scenarios, like smartphones and generic portable devices, and to evolve the idea in a permanent connection of objects to the environment [6].

The processing capabilities of every computing system are related to the use of energy in the so called "performance per Watt" metrics. Usually, when a system requires higher performances, it uses more energy, even though it is not a rigid rule. Indeed, thanks to novel effective methods and techniques for energy efficiency, the most recent computerized systems can maintain high performances for the same Watt unit [7]. It could seem that if the factor of performance per Watt does not improve over time, the electrical costs for keeping the systems in an active state would end-up much higher than the price of the hardware [8,9].

For some current applications, the problem of energy efficiency is related to the need for equipping sensor nodes with a battery. Although developments on battery capacities could be performed for increasing energy efficiency, the need for replacing low battery duration yet represents an unresolved problem [9–11]. In the worst case, the battery cannot be recharged enough. In high-performance processing systems, such as real-time Digital Signal Processing (DSP), this is a big limitation. For this reason, more and more opportunities for the use and development of IoT systems were found during these years. Several promising lines of research, related to energy harvesting [11] and passive WSNs [12], represent hot topics to the scientific community.

Due to the implementation of electronic systems in a great variability of scenarios, the need to adapt the various technologies was of primary importance. Research and development advancements concerning this problem never stop. Several issues have been overcome, including the management of the ever-increasing amount of online information. At the same time, developing different physical solutions depends on each specific problem. Programmable Logic Devices (PLDs) can tackle such a kind of problem. PLDs have been used in many contexts for years, especially when high-processing capacity is required. This issue leads to the design of intelligent re-configurable interfaces. Such a goal can be achieved by using Complex PLDs, including Field Programmable Gate Arrays (FPGAs). FPGAs allow the end-user to reprogram them several times and, thanks to their intrinsic structure and storage technologies, represent an ideal compromise between computational power and adaptability [13]. The peculiarity of this technology is represented by the ability to configure and reconfigure (even at runtime) the logical functioning of the hardware. This involves the creation of partially configurable devices, which can perform multiple functions (even independent of each other) thanks to the possibility of being able to change the configuration of some circuit portions, according to the processing requirements. In such a context, we analyze the possibility of energy efficiency [14]. The prerequisite for such a hardware interface is the hardware-software co-design, which denotes a design of the final system by means of a cooperative design process between hardware and software, which are strongly linked and adapted to each other [15].

Using new configuration storage technologies, followed by the development of hardware-specific programming languages, allows for further abstraction levels that significantly simplify the prototyping process on PLDs. Nowadays, the programming standard is represented by the family of Hardware Description Languages (HDLs), such as VHDL and Verilog. Thanks to the great capabilities of such languages, it is possible to create and test architectural hardware without having to change, modify, or physically limit the whole circuit [16,17]. In addition, high-level synthesis methods have been created to generate HDL descriptions starting from algorithms coded in software programming languages, such as C or C++. Thanks to the abstraction possibilities provided by these high-level programming languages, logic capabilities considerably overtook the classic functionality of logic devices. Besides the ability to design the entire static design of an FPGA in a simpler and faster way, by means of abstract structures, it is possible to achieve peculiar capabilities, such as partial dynamic reconfiguration. Partial reconfiguration is an advanced technique that allows one to change the configuration of a FPGA at runtime [18,19]. The dynamic reconfiguration of systems, which uses functionality to replace configuration data without interrupting its operation, has been provided for many years [20]. The ultimate goal is to keep most of the area and resources available for more than one hardware module. The design of modules that are built within the FPGA requires that the project is specifically mapped onto the internal hardware [21]. Reusing the same partition allows for a virtual extension of the chip area. Even though not every FPGA device supports partial reconfiguration, Xilinx® Inc. (San Jose, CA, USA) has produced several systems that fully support this feature. The complex process of development and control required during runtime limits the feasibility of this technique in many real-world applications.

Xilinx released the Vivado Design Suite that supports both static and partial programming, thus providing a variety of automatic tools and facilities that help the designer in all design phases [22].

Despite the basic flow for static programming, some steps could be added to design a adaptive system using a dynamic partial reconfiguration technique: a chip region has to be prefigured to make it reconfigurable, setting some parameters by using multiple design implementations. The final product is the total bitstream creation for booting and partial configuration for each module that is to be replaced at runtime. The designer can change this workflow, regardless of the physical structure of the FPGA.

For the development of the proposed DSP system, ZedBoard was used, a board produced by Xilinx in collaboration with Digilent Inc. (Pullman, WA, USA). The on-board chip belongs to the Zynq-7000 family of All Programmable System on Chip (AP SoC), which integrates one of the latest generation of FPGAs in the Programmable Logic (PL) with a dual-core ARM Cortex-A9 (Cambridge, UK) microprocessor in the Processing System (PS) into a single integrated circuit. Its use is especially suitable for applications that require software/hardware functionality. This innovative system enables the analysis of the impact of the use of partial reconfiguration on energy consumption.

The proposed work is based on the automated DSP of video data streams yielded by a video pattern generator. Filtering can be performed at software-level, as well as at hardware-level, using three different accelerators in the same reconfigurable portion. FPGAs have been designed to generate three pipelines for video stream processing: (i) one for capture, (ii) one for filtering, and (iii) the last one for viewing. At software-level, the CPU controls the data-flow by managing the transfers among the pipelines and Double Data Rate (DDR) memory. It also performs the dynamic loading of the modules that exploit partial reconfiguration. We used the Vivado Design Suite (2017.1, Xilinx Inc., San Jose, CA, USA) during all the steps that led to the final creation of both total and partial bitstreams. A special device configuration unit equipped with the ARM processor allowed for the loading of the partial configuration bitstreams through the Device Configuration (DevC) and Processor Configuration Access Port (PCAP) interfaces. The used operating system is based on Linux and runs on the ARM microprocessor, while the used kernel and the boot loader are provided by Digilent.

The final results show that partial reconfiguration can lead to significant energy efficiency advantages, which are mainly related to low software performances and the idle times of hardware modules.

The paper is structured as follows. Section 2 outlines the background; Section 3 analyzes the development environment; Section 4 describes the proposed reconfigurable system; a case study on DSP is presented in Section 5; Section 6 introduces the analytical framework for energy efficiency evaluation, as well as showing and discussing the experimental results; finally, some discussions and conclusive remarks are given in Sections 7 and 8, respectively.

2. Related Works

Dynamic Partial Reconfiguration (DPR) allows one to modify several modules in a static design on the FPGA device and can potentially reduce the number of devices or the device size, thereby reducing both size and power consumption. To date, there have been several works about the partially and dynamically reconfigurable systems. Some of these works have mainly introduced a simple reconfigurable system and focus on the advantages of the proposed dynamic partial reconfiguration design flow. This design type could be exploited in many application fields, for example to meet space requirements in small portable systems, as well as to create a system-on-a-chip with a very high-level of flexibility [23].

The study on adaptive allocation of limited FPGA resources is also applicable to hardware-accelerated software-defined radios. A system that requires one to either transmit or receive capabilities at any given time, but not both contextually, can switch between the two modes in a fraction of a second using partial reconfiguration. This technology considerably reduces power consumption, which represents a critical issue in portable ground-based applications. A software-defined radio was designed with a reprogrammable Forward Error Correction (FEC) block supporting multiple codecs [24].

Another work proposes four different techniques to perform DPR, namely: SelectMAP, Serial mode, Joint Test Action Group (JTAG), and Internal Configuration Access Port (ICAP). In the study, each of these techniques is reviewed, evaluated, and tested using a convolutional encoder, i.e., an essential block from a Software Defined Radio (SDR) system [25].

These architectures generally use a reconfiguration controller for scheduling and allocating total or partial configuration files, named bitstreams. Depending on the complexity of runtime reconfigurations, this controller can be either a simple finite state machine or a microprocessor. The DPR controllers provided by the FPGA vendors rely on software to manage the reconfiguration process. This approach may lead to slow reconfiguration and unpredictable timing. So, an alternative approach was developed for designing an open-source DPR controller specialized for real-time systems. The controller enables a processor to perform reconfiguration in a time-predictable manner and supports different operating modes [26].

Regarding energy consumption in digital systems, there are two main types of energies that are used: (i) static energy and (ii) dynamic energy. Static current consumption occurs due to intrinsic losses of the transistors, while the dynamic current is used during transistor state switching. A recent study released by Xilinx indicates that, below 0.25 μm (referring to the semiconductor production process), the static energy consumption exponentially increases with every new production process [27]. This study states that the static component is becoming the largest percentage of the total energy consumption.

The analyses on energy losses of 90 nm FPGAs were performed using detailed device-level simulations. Especially, static energy consumption was found to be directly dependent on the configuration bit values. In addition, this work indicated that the polarity of the inputs/outputs of the circuits has a strong impact on the current consumption. Especially, the authors indicated that in the modern commercial process that uses the CMOS technology, the energy consumption from the elementary structures that compose FPGA devices, such as buffers and multiplexers, is significantly lower when their outputs and inputs are configured in logic 1 versus logic 0 [27].

On the other hand, the dynamic energy consumption must be also taken into account. Several studies focused on the impact of the clock signal on the overall chip energy consumption. The achieved results show that the clock distribution can contribute up to 22% of the total power. To reduce the consumption of these lines, various solutions were developed, such as clock gating [28], which allows one to turn certain circuit clocks on or off, or enables clock frequency reduction [29].

A more interesting breakthrough in these studies implies the use of reconfiguration at runtime. In the past, circuits were proposed in which particular functions were bound to specific regions, allowing for turning-off unnecessary components at runtime [29]. This class of schemes often requires modification of the FPGA architectures and hardware implementation.

In this work, we will concentrate on energy efficiency techniques that could be applied to existing and commercially available partial dynamically reconfigurable FPGAs, such as Xilinx (all the families of the 7 Series) and Altera Corp. (San Jose, CA, USA) products. In particular, the case of the potential of partial reconfiguration will be treated and analyzed, in order to achieve the maximum balance between computing and dissipated power.

DPR is mainly used to realize adaptive hardware in a dynamic environment. Firstly, it speeds-up hardware algorithms that may be particularly burdensome when running on software platforms. Secondly, it allows one to efficiently use the chip area, so that several hardware modules can be interchanged while the rest of the system continues its execution. Finally, it is possible to implement an energy-saving policy by replacing the inactive modules with other ones that virtually do not dissipate power.

3. The Development Environment

This section describes the development environment by firstly introducing the Vivado Design Suite, followed by the ZedBoard development board.

3.1. The Vivado Design Suite

The Vivado Design Suite is a design environment developed by Xilinx to increase the overall productivity for design, integration, and deployment on Xilinx 7 Series and UltraScale FPGA platforms. The need for a unique automated environment arises from the evolution of ever more complex System on Chip (SoC) devices, which introduce a different approach in the programming phase. These new devices give rise to multidimensional design challenges when they are handled incorrectly, heavily affecting faster development times and greater productivity. A scalable and sharable data model is used, so that the entire design process can be run in memory without requiring one to write or translate intermediate file formats, which would slow-down implementation, debugging, deployment phases, and even increase memory requirements [30].

All the Vivado Design Suite tools are written with the native scripting Tool Command Language (Tcl), which can be used with a command interface. A Tcl script can cover the entire design and implementation stream, including all the needed reports generated for design analysis at any point in the design process [30]. The main advantage is the complete control over each whole workflow [31].

3.2. The ZedBoard Development Board

The peculiarity of the ZedBoard development board lies in the Xilinx chip of the Zynq-7000 family, which includes a configurable logic part, called PL and a PS [32,33]. The Zynq-7000 family is based on the Xilinx AP SoC architecture that defines the modern standards for embedded systems. This type of integrated chip is based on a single processor (Zynq-7000S) or a dual (Zynq-7000) ARM Cortex-A9 MPCore core that manages both the PS part and an FPGA that performs the PL, in a single device [34]. This chip is built with state-of-the-art processes, and high-performance and low-power technologies, using a 28 nm integration and High-K Metal Gate (HKMG) transistors [35,36].

The typical architecture of these circuits allows for easy mapping of customized logic and software in PL and PS, respectively, so that it is possible to create unique and uncommon features with respect to any other system. Combining PS and PL provides performance levels that two-chip solutions (such as a CPU and a separate FPGA) cannot match due to their limited I/O bandwidth, low-coupling capacity, and power.

4. The Proposed Reconfigurable System

The system proposed and analyzed in this work is a combination of hardware and software design, developed and implemented using tools, techniques, and components provided by Xilinx and Xylon. One of the main objectives of the paper is the development of a general dynamically and partially reconfigurable infrastructure for the energy efficiency evaluation in dynamic, partially reconfigurable FPGA devices. More specifically, the design methodology for the implemented DSP application was adapted for the ZedBoard device.

This section is divided into two parts: the former explains how a partial reconfiguration hardware design on Vivado Design Suite is created; the latter deals with the software guidelines, explaining the dynamical part of partial reconfiguration process.

4.1. Design Workflow and Implementation

The design workflow for the latest generation systems, like the 7000 Series chip, is developed on the Vivado software suite, which provides a new paradigm of hardware designing. Such software can support the automation of low-level detail management to meet the requirements of the different supported chips. In general, the user should only provide directions for defining the design structure and dealing with floor-planning. The partially reconfigurable design process is similar to the standard stream, but with some additional steps. This particular design flow requires the implementation of multiple configurations that ultimately translate into the creation of total bitstreams, for each configuration, and partial bitstreams, for each reconfigurable module. The amount of the required configurations varies according to the number of modules that need to be implemented. However, all configurations use the same static and routed design, exporting it from the initial configuration and importing it into all the subsequent ones. The detailed description of processing steps involved in a partial reconfiguration project can be retrieved from [22].

The main contribution of this work regards the development of an infrastructure for the partial dynamic reconfiguration technique. In our case, we used the ZedBoard as target device for our implementation. The design workflow was adapted to consider the common elements and steps to support our analysis. Although the implementation steps can be applied regardless of the target device, thanks also to the Vivado automation processes, the first design step, which is the creation of a block diagram starting from the combination of IP cores, needs to be adapted to each target system. Some IP cores represent specific feature of the target board (e.g., the PS IP Wrapper) and cannot be employed in general design processes. Therefore, the first phase involved the creation of a diagram, wherein the DSP blocks (i.e., video capturing, filtering, and visualization) represented the starting point of the design. Then, the specific blocks of PS, clock, interrupt, and interconnections were added to the make the whole design specialized to the target device. Consequently, device-dependent constraint files were created to bind the design signal line to the board's physical pins and imported in Vivado. Generally, the constraint files are used during subsequent design phases.

After choosing a reconfigurable target block, the design flow is characterized by synthesis and implementation processes, which are based on the static design flow. The main difference concerns the outline of the portion of the target chip (in our case the PL of the Zynq-7000) that needs to be reconfigurable and starts the process performed by Vivado for the final creation of the partial configuration files.

4.2. Software Guidelines

The configuration process of dynamic partially reconfigurable systems follows the same steps as the classical approach:

- After the power-on reset, the boot ROM determines the external memory interface, the boot mode, and encryption status. The ROM uses the DevC Direct Memory Access (DMA) to load the First Stage Boot Loader (FSBL) into the Integrated RAM memory;
- The FSBL control is released to the CPU, which configures the PL with the static design bitstream using the PCAP port. The device is now completely configured and working;
- The FSBL loads and releases control at the second boot loader (U-boot) that loads the image of the Linux kernel, the binary file tree, and the Linux system root, and finally releases the control to the Linux kernel;
- At the end of the boot process, by means of a Bash script, a Linux application is started.

From this moment, the application can use partial bitstreams to modify the logic circuit in the reconfigurable portion, while the rest of the FPGA continues to run. This is accomplished by transferring the partial bitstream from the DDR to the PL through the PCAP. A single configuration engine handles the full configuration, as well as the partial reconfiguration.

The task of loading a partial bitstream in the PL does not require knowledge of the physical location of the reconfigurable module, since the configuration frame address information is included in the partial bitstream. So, it is not possible to dynamically assign the location where the reconfigurable module has to be loaded, because the location is bound with the module during design and implementation steps on Vivado.

At the application-level, reconfiguration is enabled and managed by the *xdevcfg* driver provided by Digilent. This driver allows one to perform a complete configuration or partial reconfiguration of the PL. It is built on a virtual file system, where the reconfiguration is performed by writing the bitstream into a certain location of the Linux file system. This operation activates the DMA transfer and blocks the operating system by polling, awaiting an event that indicates the end of the reconfiguration phase. The reconfigurable region is considered as a reconfigurable peripheral, where the only fixed parameter is its range of memory allocations. This peripheral is connected to the ARM via an Advanced eXtensible Interface (AXI) bus interface, while partial bitstreams are already generated and stored on the SD Card mounted onto the Linux file system [37].

5. Case Study: Video Filtering

In our DSP application, the video filtering features are dynamically modified. The original video stream generated automatically by the Test Pattern Generator (TGP) block is processed by three different filters (Figure 1), and is developed from an algorithmic description in C language using the high-level synthesis software Vivado HLS to generate three different filtering IP cores:

- Posterize [38], which is a filter that is applicable to an image and yields a compressed image. The bit-depth regarding color levels is reduced, while the contrast is increased. The posterized image is less heavy in terms of file size, but it might be subject to quality degradation. The resulting effect could remind the comics or posters;
- The Sobel operator [39], which is a discrete high-pass filtering technique used to process digital gray-level images to detect edges and transitions. Usually, the result approximates the gradient of the image intensity function, conveying the information for contour recognition;
- The Features from Accelerated Segment Test (FAST) algorithm [40] is generally defined as a method for recognizing angles in images. It is used to extract information about tracking and mapping objects in several computer vision applications.

As pointed out before, the pre-compiled IP core implementations available on the Xilinx website were used. The hardware and software versions of the FPGA implemented filtering methods have the same features, in terms of data structure and numeric precision, as algorithms with comparable instructions and operations. The used codes and IPs were generated using the Vivado HLS with default settings, in order to have a comparable degree of optimization in the analyzed software and hardware filter implementations. In this way, energy efficiency evaluations were not affected by generation quality/optimality issues, and a direct comparison between the hardware and software implementations is feasible [41].

Figure 1 shows the three video filtering modes applied on a video streaming pattern, dynamically generated by the TGP block. Note that this block can generate different types of video patterns. In order to highlight the filter functioning, a dynamic pattern was chosen. The pattern consists of a matrix of gradient squares that represents the background and a small green square object that moves throughout the screen.

5.1. Hardware Implementation

In order to reconfigure the PL using a bitstream, the FPGA configuration memory must be accessed. Xilinx FPGAs offer the following communication interfaces: JTAG, SelectMAP, and ICAP. JTAG and SelectMAP can be accessed from outside the FPGA and require an external reconfiguration controller. On the contrary, ICAP can be accessed from within the FPGA, which allows for self-reconfiguration.

Significant effort has been devoted on designing new interface structures for ICAP to speed-up performance, as well as to reduce the required resources [42–45]. Studies regarding these new approaches led to the development of the PCAP reconfiguration interface. This interface is available on the most recent Xilinx devices, such as the Zynq-7000 family. PCAP resides in the PL and enables the reconfiguration of the FPGA using the ARM processor through its device configuration module, called DevC, which is located in the PS.

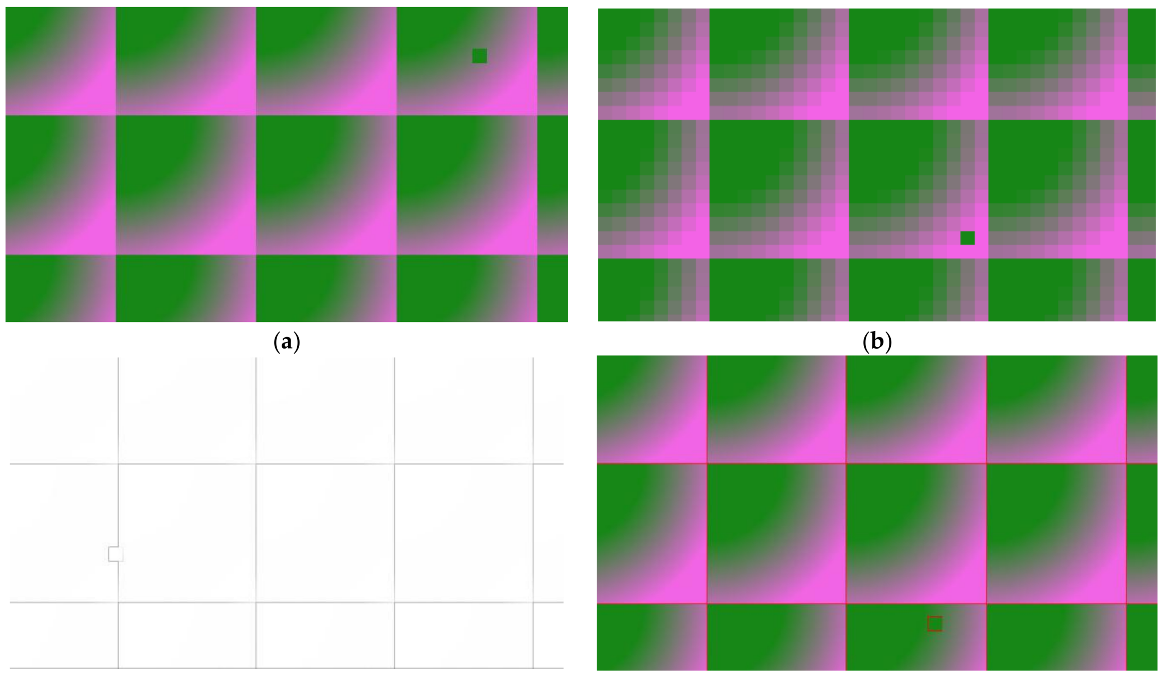

(**a**) (**b**)
(**c**) (**d**)

Figure 1. The three hardware filters working on an instance of the video stream generated by the TGP block: (**a**) original video, (**b**) posterize, (**c**) Sobel operator, and (**d**) FAST. Note the small green square object moving throughout the display.

Such a kind of design uses both the PS and PL parts and reveals how the control section (mapped onto the PS) and the data path (mapped onto the PL) can be separated. FPGA devices are implementing a powerful high-definition digital video processing circuit, consisting of a capture pipeline, a memory-to-memory processing pipeline, and a video output pipeline. The PS is used to run a Linux-based operating system, called PetaLinux, in which a software performs partial reconfiguration of the modules in the PL (see Figure 2) [37].

Partial bitstreams are transferred to the DevC via the central bus of the ARM, and a built-in DMA is used to speed-up this process. On the other hand, DevC is connected to the PCAP of the PL. The DevC block incorporates an AXI/PCAP bridge, which helps to convert messages from the AXI bus to those compatible with PCAP and vice versa. Receiving and transmission FIFO queues are used in both directions of communication to move reconfiguration data between the different domains of the PCAP and AXI.

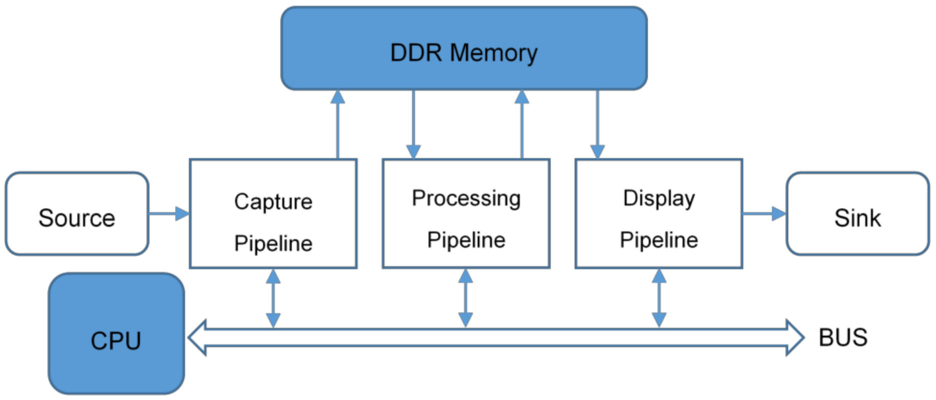


Figure 2. The data-flow pipeline control performing partial reconfiguration of the hardware modules [37].

5.2. Case Study Implementation

The implementation of the design previously described complies with the same procedures mentioned in Section 4.1. We can focus on the main features of our case study:

- For the creation of the block diagram, some default blocks were exploited from the Vivado IP core catalog, and others imported from third parties to achieve the functionality required by the design. Figure 3 shows a simplified version of the block diagram (whose functional elements were implemented in the Vivado IP environment), while Figure 4 depicts the 'processing' block, which is the high-level module that contains the 'image filter' IP core that must be set as a reconfigurable block;
- Figure 5 shows the physical resources used in the chip after the implementation process. The main logic of the static design is marked in blue; the area occupied by the PS in orange and the pblock_image_filter_1 is empty, because the black box was loaded after the implementation (purple outline). The areas marked by white squares are the partition pins, i.e., physical interfaces between static and reconfigurable logic. They are anchoring points of the module within a reconfigurable portion and allow for the interconnection of each module I/O with the static part interface paths;
- After the implementation process of the tree filters (Figure 6), the system is ready to be translated into configuration files.

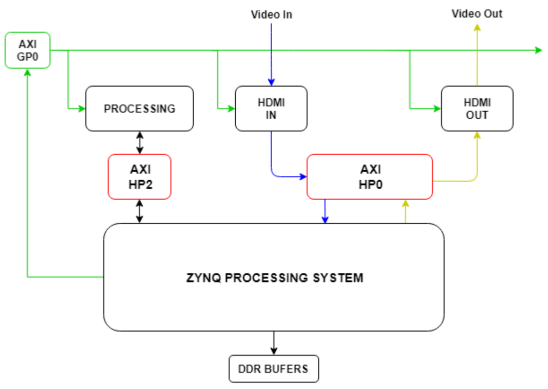


Figure 3. Simplified block diagram of the main IP cores used for the entire design. The green connections represent the configuration and control of the IP cores; blue and yellow colors denote the video input and output lines, respectively; the two red blocks handle the memory buffers communication through the Zynq PS.

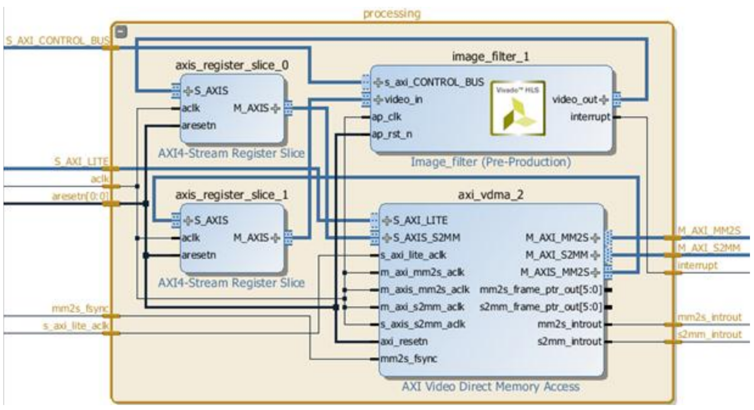


Figure 4. Detailed scheme of the 'processing' block, showing the internal organization that is composed of: two AXI4-Stream Register Slice cores, which are multi-purpose pipeline registers that are able to isolate timing paths between master and slave; an AXI Video Direct Memory Access that provides high-bandwidth direct memory access between memory and AXI4-Stream video type target peripherals; and an 'image filter' IP core (generated by the Vivado HLS) that performs the video processing. The 'image filter' IP core is the only reconfigurable block.

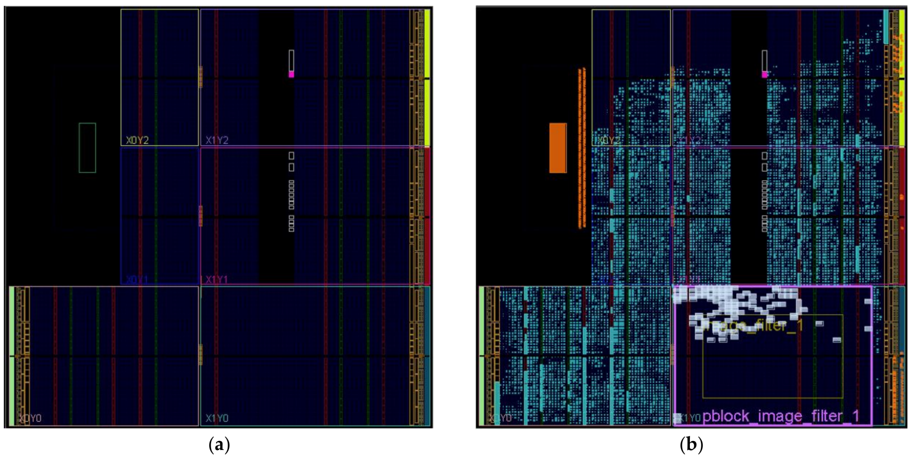


Figure 5. The representation of the chip's resources utilization: before (**a**) and after (**b**) the implementation process.

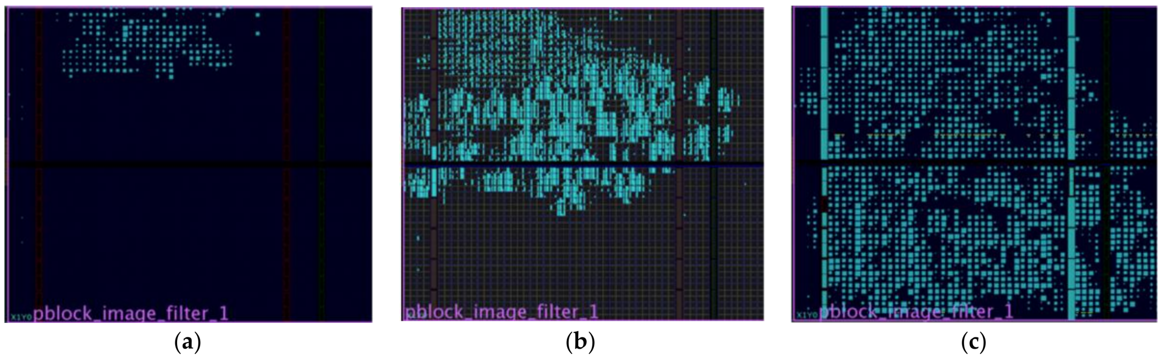


Figure 6. The desired reconfigurable module for the three investigated filters: (**a**) posterize, (**b**) Sobel operator, and (**c**) FAST.

6. Experimental Results

6.1. The Analytical Framework

An analytical framework allows one to understand and analyze physical measurements. In this section, we analyze how and under which conditions partial dynamic reconfiguration can save energy.

The energy consumption in electronic devices depends on structural specifications (i.e., used technology, semiconductor production process, working frequency), as well as on functional properties (i.e., DSP task and technique, coding language, requested throughput). Let $\mathcal{P}_s$ and $\mathcal{P}_f$ be the sets of structural and functional properties, respectively. Moreover, reconfigurable FPGAs introduce an additional level of complexity. As a matter of fact, FPGAs can dynamically change the DSP functionality by keeping constant the structural features [18]. Accordingly, the gap between $\mathcal{P}_s$ and $\mathcal{P}_f$ is narrowed, because the same hardware resources can be exploited for different applications (modifying configuration and routing). Therefore, energy consumption depends not only on $\mathcal{P}_s$ and $\mathcal{P}_f$ but also on a set of parameters $\mathcal{P}_{\text{FPGA}}$ that is tightly related to each reconfigurable hardware architecture. $\mathcal{P}_{\text{FPGA}}$ could include reconfiguration time and working frequency assigned to a specific module.

Therefore, an analytical model that exhaustively assesses how much each parameter affects energy consumption is very difficult to accomplish, due to the huge number of parameters that need to be simultaneously considered. In addition, some of these parameters cannot be quantitatively measured but just estimated. More specifically, our study is mainly focused on the relationship between the dynamic partial reconfiguration time $t(r)$ and the corresponding energy consumption regarding the static/dynamic architectures [11]. The used evaluation metrics are described in what follows.

6.2. Energy Metrics

Let $E(\text{hw})$ be the energy that is consumed by the hardware module and $E(\text{sw})$ the energy consumed by a CPU to carry out the same task via software; to this aim, the constraint in Equation (1) must be used to determine under which conditions an energy-saving is actually achieved [46]:

$$E(\text{hw}) < E(\text{sw}), \tag{1}$$

considering that the energy is the product between power and processing time (by considering the average power), we obtain:

$$P(\text{hw}) \cdot t(\text{hw}) < P(\text{sw}) \cdot t(\text{sw}), \tag{2}$$

in which $P(\text{hw})$ and $t(\text{hw})$ indicate the power and the execution time concerning the hardware module, respectively, while $P(\text{sw})$ and $t(\text{sw})$ denote the power and the execution time of the software implementation, respectively. Relying on Equations (1) and (2), two additional metrics can be calculated:

the ratio between $P(\mathrm{hw})$ and $P(\mathrm{sw})$ is the power-up (P_{up}) parameter, while the ratio between $t(\mathrm{sw})$ and $t(\mathrm{hw})$ is the speed-up parameter (S_{up}):

$$P_{up} = \frac{P(\mathrm{hw})}{P(\mathrm{sw})}, \tag{3}$$

$$S_{up} = \frac{t(\mathrm{sw})}{t(\mathrm{hw})}. \tag{4}$$

Accordingly, Equation (2) becomes:

$$\frac{P_{up}}{S_{up}} < 1. \tag{5}$$

Equation (5) allows one to understand how the energy efficiency of the hardware accelerator is only possible if this ratio is less than 1. This applies when the module is instantiated and is running as a single circuit. Considering the idea of using the partial dynamic reconfiguration to offload the accelerator's region when it is not active, both static and dynamic energy can be reduced [47]. In our experimental trials, we used an empty module, called black box, to offload the reconfigurable region during the idle time. This constraint occurs because there is not any circuit that consumes a lower quantity of energy with respect to one that physically does not exist.

However, the reconfiguration operation requires an energy overhead caused by the reconfiguration process itself. It is possible to analyze this situation by a simple relationship that links the additional energy required to reconfigure the module $E(r_{\mathrm{module}})$ with the saved energy by downloading the module $E(\mathrm{blackbox})$:

$$E(r_{\mathrm{module}}) < E(\mathrm{blackbox}). \tag{6}$$

Moreover, in this case every term can be transformed in terms of the power consumed over time (considering the average power):

$$P(r_{\mathrm{module}})\cdot t(r_{\mathrm{module}}) < P(\mathrm{blackbox})\cdot t(r_{\mathrm{idle}}), \tag{7}$$

in which $P(r_{\mathrm{module}})$ and $t(r_{\mathrm{module}})$ are the power and time needed for the reconfiguration, respectively, while $P(\mathrm{blackbox})$ and $t(r_{\mathrm{idle}})$ are the power of the downloaded hardware module (i.e., the power consumed by charging the black box during the idle period) and the time in the idle period, respectively.

It is worth noting that during the partial reconfiguration, a bitstream configuration file is sent to the PCAP port, which writes the configuration data into the configuration memory. Therefore, the parameter $P(r_{\mathrm{module}})$ depends on the architecture and hardware of the chip, while the parameter $t(r_{\mathrm{idle}})$ depends on the application. Unlike the other parameters, the reconfiguration time $t(r_{\mathrm{module}})$ is purely a function of the bitstream file size. The time increases fairly linearly by the same amount of data received, with minimal latency variance depending on the location and content. In general, it is possible to define this relationship:

$$t(r_{\mathrm{module}}) = \frac{D(\mathrm{bits})}{T(\mathrm{PCAP})}, \tag{8}$$

in which $D(\mathrm{bits})$ and $T(\mathrm{PCAP})$ are the dimension of the bitstream and the throughput of PCAP, respectively. The 32-bit PCAP interface of the Zynq-7000 has a 100 MHz clock and supports a maximum download throughput of 400 MBps.

Since the variables dependent on the hardware can be measured, i.e., $P(\mathrm{blackbox})$ and $P(r_{\mathrm{module}})$, as well as $t(r_{\mathrm{idle}})$, this reduces the overhead of energy due to reconfiguration results in an increase of the configuration data transfer throughput (in the case of ZedBoard, it is set to its maximum capacity).

Using Equation (9), produced by replacing Equation (8) with (7), it is possible to establish the lower bound for introducing energy-savings:

$$\frac{P(r_{\text{module}}) \cdot D(\text{bits})}{P(\text{blackbox}) \cdot t(r_{\text{idle}}) \cdot T(\text{PCAP})} < 1. \tag{9}$$

Equations (1) and (6) are useful to evaluate the conditions that lead to energy efficiency in terms of hardware against software processing and the energy consumption introduced by the reconfiguration process, respectively. In this work, we aim to investigate energy efficiency in the case of a partial dynamic reconfiguration system against the static version. Therefore, we introduce an original formulation that coherently extends the metrics presented in [46,47]. To the best of our knowledge, we represent for the first time the specific case of energy efficiency of dynamic partial reconfiguration with respect to non-reconfigurable devices in Equation (10):

$$E(\text{reconfigurable}) < E(\text{static}), \tag{10}$$

end expanding it into:

$$E(r_{\text{active}}) + E(r_{\text{module}}) + E(\text{blackbox}) < E(s_{\text{active}}) + E(s_{\text{idle}}), \tag{11}$$

in which

- $E(r_{\text{active}})$ is the energy of the hardware module when it is running;
- $E(\text{blackbox})$ and $E(r_{\text{module}})$ are the energy saved by downloading the module and the energy required for the reconfiguration process, respectively;
- $E(s_{\text{idle}})$ refers to the energy used while keeping the module during its idle static period.

Since the energy required for an active operation is the same in both configurations (i.e., $E(r_{\text{active}}) = E(s_{\text{active}})$), by replacing the energy with the power over time, we can define Equation (12):

$$P(r_{\text{module}}) \cdot t(r_{\text{module}}) + P(\text{blackbox}) \cdot t(r_{\text{idle}}) < P(s_{\text{idle}}) \cdot t(s_{\text{idle}}). \tag{12}$$

The idle time $t(s_{\text{idle}})$ must be equal to the reconfiguration time $t(r_{\text{module}})$ added to the idle time of black box $t(r_{\text{idle}})$, because we aim to compare the two systems in an equal time period. Making the same considerations for previous Equations, and manipulating (12), leads to:

$$\frac{D(\text{bits}) \cdot [P(r_{\text{module}}) - P(s_{\text{idle}})]}{t(r_{\text{idle}}) \cdot T(\text{PCAP}) \cdot [P(s_{\text{idle}}) - P(\text{blackbox})]} < 1. \tag{13}$$

Equations (5), (9), and (13) are used to find under which conditions the partial reconfiguration can provide remarkable energy efficiency. Therefore, the new framework allows for a direct comparison between the energy efficiency achieved by a dynamic partially reconfigurable device with respect to a static non-reconfigurable system.

6.3. Power Consumption Measurements

Due to the fact that the ZedBoard has not multiple power sources, a digital oscilloscope to measure the voltage V_{J21} across the 10 mΩ shunt resistor R_S (performed using pins of the J_{21} current sense connector). As in the Equation (14), by taking the V_{J21}, dividing by the resistance R_S and multiplying it by the power supply voltage V_{PS} (in our case, the nominal value is 12 V), it is possible to derive the total power P dissipated by the board.

$$P(\text{W}) = \frac{V_{\text{J21}}(\text{V}) \cdot V_{\text{PS}}(\text{V})}{R_S(\Omega)}. \tag{14}$$

As expected, the achieved values are in line with our hypotheses, with a significant increase in the configuration with the FAST filter compared to the one with the video in OFF mode. For statistical significance, 15 different repetitions for each different measurement were performed, and we calculated the mean value and the standard deviation in Table 1.

Table 1. The software/hardware total power consumption (average and standard deviation values) of the proposed configurations.

Configuration	**Total Power/*P* (W)**			
-	**Software**		**Hardware**	
-	**Avg.**	**Std. Dev.**	**Avg.**	**Std. Dev.**
Posterize	3.876	1.710×10^{-2}	3.960	1.633×10^{-2}
Sobel operator	3.900	1.749×10^{-2}	4.020	1.743×10^{-2}
FAST	3.912	1.787×10^{-2}	4.680	1.673×10^{-2}
Video ON	3.674	1.811×10^{-2}	3.674	1.811×10^{-2}
Black box	3.380	1.587×10^{-2}	3.380	1.587×10^{-2}
Video OFF	2.280	1.936×10^{-2}	2.280	1.936×10^{-2}

It is worth noting the difference between the hardware filters and the software version. This gap is mainly due to the filtering system: for 'hardware', we mean the filtering operation performed by the FPGA, while for 'software' we mean the software filter executed by the CPU. The hardware accelerator that runs on the FPGA consumes slightly more energy with respect to the CPU, because the dedicated hardware system generally allows for a remarkably higher throughput in the specific application.

With reference to the Video ON and black box configurations, the first one concerns the display of an unfiltered video with one of the three filters loaded but disabled, and the second one displays an unfiltered video, loading the black box in place of the filters.

The evaluation can be completed starting from Equation (5). We considered the Watt consumption in Table 1 to calculate the P_{up}, and the time to process a frame required by the software filter and the hardware filter for the S_{up}. These metrics were experimentally measured for each different filter. The hardware filter supports 60 frames per second (fps), so it processes a frame every 1/60 s (approximately 0.017 s), while the software filter can process at 2 fps, so it processes a frame every 0.5 s. The obtained values are reported in Table 2.

Table 2. The average energy efficiency of the proposed configurations.

Configuration	**Power-Up (P_{up})**	**Speed-Up (S_{up})**	**Saved Energy (S_E) (%)**
posterize	1.021	29.410	96.6
Sobel operator	1.030	29.412	96.5
FAST	1.196	29.254	96.0

We can argue that the relationship between P_{up} and S_{up}, called saving factor (S_f), is significantly less than 1; thus, in our case a great energy efficiency, associated with the hardware acceleration of the filters, is achieved. The saving factor indicates the amount of energy that is required by the hardware version as the software version to perform the same operation. As a result, considering that the Saved Energy is $S_E = 100\left(1 - S_f\right)$ with a saving factor of 1 (i.e., $S_E = 0\%$), both systems are equivalent at 0.5 (i.e., $S_E = 50\%$), which means that the hardware implementation requires half of energy consumed by the corresponding software implementations. Table 2 shows the savings in percentage calculated for each filter. Note that low software performances are due to the heavy Full HD video processing that the CPU needs to run, which severely drops the factor $t(\text{sw})$ and actually increases the corresponding S_{up} value.

As in Equation (9), it is possible to isolate known or derived parameters and express the equation in function of a single variable. The requested energy for the reconfiguration was calculated by measuring the total power used during the reconfiguration time of the three filters. On average, this process consumes 1.5 W more than the value of the 'Video OFF' condition; thus, the value of $P(r_{\text{module}})$ is 3.69 W for posterize, 3.78 W for sobel operator, and 3.77 W for FAST filter. This value is justified, because the capture and DSP pipelines are disabled at software-level (before loading the new filter). The value of $P(\text{blackbox})$ can be taken from Table 1, while the size of the partial configuration .bit file does not change, because it must respect a default format ($D(\text{bits}) = 736$ KBps). The parameter indicates that the inactivity of the module $t(r_{\text{idle}})$ can be left as a variable to detect under which conditions energy efficiency can be achieved by hardware accelerator inactivity. Replacing these values in Equation (9), we can evaluate the minimum time $t(r_{\text{idle}})$ that is necessary for energy saving despite the additional energy consumed by the reconfiguration process, which allows one to achieve the desired efficiency conditions (Table 3).

Table 3. The energy efficiency conditions when reconfiguration process does not nullify the energy savings that would be achieved with the reconfiguration. The value $t(r_{\text{idle}})_{\text{Saving}(\%)}$ indicates the minimum time required by the black box to save a quantity of energy equal to the saving factor in percentage.

$E(r_{\text{module}}) < E(\text{blackbox})$			
$t(r_{\text{idle}})_{SE(\%)}$ (ms)			
$t(r_{\text{idle}})_{25\%}$	**$t(r_{\text{idle}})_{50\%}$**	**$t(r_{\text{idle}})_{75\%}$**	**$t(r_{\text{idle}})_{99.9\%}$**
2.750	4.120	8.240	2060

Equation (13) allows to obtain the results regarding the case of energy efficiency of reconfigurable architectures against the static ones. As before, it is possible to achieve the minimum time by bringing the factor $t(r_{\text{idle}})$ to the second member in Equation (13). From Table 1, the Value of 'Video ON' was selected for the variable $P(s_{\text{idle}})$, since it is the value of the module loaded but still not in operation. By replacing the variable values in Equation (13), we obtain $t(r_{\text{idle}})_{\text{min}} > 0.663$ ms.

By following the same procedure described above, it is possible to obtain the energy efficiency metrics by referring to the reconfigurable case against the static one. The four achieved values represent the minimum idle time that is necessary for the black box to obtain four different energy-saving percentages. The value $t(r_{\text{idle}})_{SE(\%)}$, which indicates the idle time that is required to save a quantity of energy that is equal to the saving factor in percentage must not be less than a few milliseconds to achieve a noticeable advantage. Comparing the reconfigurable architecture with the static one, it is possible to deduce that the idle time that was necessary to realize notable energy efficiency is always in the order of a few milliseconds (Table 4).

Table 4. The energy-saving ratios that refer to the reconfigurable case against the static version. The value $t(r_{\text{idle}})_{SE(\%)}$ indicates the minimum time required by the black box to save a quantity of energy equal to the saving factor in percentage.

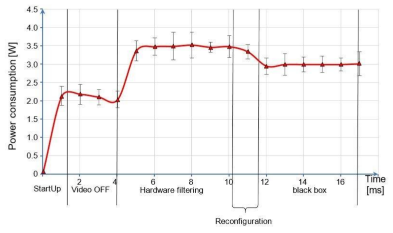
$E(\text{reconfigurable}) < E(\text{static})$			
$t(r_{\text{idle}})_{SE(\%)}$ (ms)			
$t(r_{\text{idle}})_{25\%}$	**$t(r_{\text{idle}})_{50\%}$**	**$t(r_{\text{idle}})_{75\%}$**	**$t(r_{\text{idle}})_{99.9\%}$**
0.884	1.326	2.652	663

Table 5 reports the average values with the corresponding standard deviation measured (over 15 repetitions) by means of a digital oscilloscope using a sampling time of 1 ms. Figures 7–9 show the energy consumption, expressed in Watts versus time, concerning the two different architectures of the three implemented filters: partially reconfigurable and static implementations, respectively. These power consumption versus time plot curves (obtained by interpolating these real measurements using cubic Bézier splines) show measurements acquired using the J21 component (current sense) in the ZedBoard.

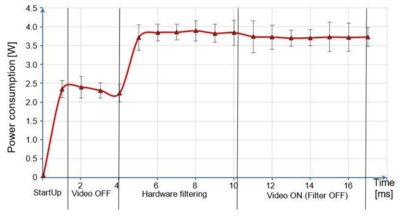
As it can be seen, the partially reconfigurable architecture can increase energy efficiency compared with the static architecture. Until t = 10 ms, the behavior of the two systems (dynamic and static) is the same; in the next times, the static configuration turns off the filter, while the dynamic one "discharges" it and loads the black box. Both perform the same initial stages of start-up, video OFF, and filtering. The power consumption data were taken from Table 1 for the values of Video OFF, Hardware filtering, Video ON, and black box, while for the reconfiguration process the average values are: 3.69 W for posterize, 3.78 W for the Sobel operator, and 3.77 W for FAST filter. For each time instant, we replicated the measurement 15 times to achieve higher robustness and reproducibility. Only the reconfiguration time (for the reconfigurable architecture) and the filter shutdown (for the static architecture) depend on the device. The former is obtained analytically from Equation (8), while the latter was measured and takes only a fraction of the configuration time.

Table 5. The power consumption for the three filters in the static and dynamic configurations. The values are expressed as average value ± standard deviation over 15 measurement repetitions.

Time (ms)	Posterize Filter Consumption (W)		Sobel Filter Consumption (W)		FAST Filter Consumption (W)	
	Static	Dynamic	Static	Dynamic	Static	Dynamic
1	$2.33 \pm 2.27 \times 10^{-2}$	$2.33 \pm 2.98 \times 10^{-2}$	$2.28 \pm 2.25 \times 10^{-2}$	$2.28 \pm 2.50 \times 10^{-2}$	$2.22 \pm 2.68 \times 10^{-2}$	$2.22 \pm 2.39 \times 10^{-2}$
2	$2.38 \pm 2.92 \times 10^{-2}$	$2.38 \pm 3.11 \times 10^{-2}$	$2.35 \pm 2.27 \times 10^{-2}$	$2.35 \pm 2.20 \times 10^{-2}$	$2.37 \pm 2.44 \times 10^{-2}$	$2.37 \pm 2.46 \times 10^{-2}$
3	$2.29 \pm 2.11 \times 10^{-2}$	$2.29 \pm 2.40 \times 10^{-2}$	$2.29 \pm 3.52 \times 10^{-2}$	$2.29 \pm 2.33 \times 10^{-2}$	$2.22 \pm 2.70 \times 10^{-2}$	$2.22 \pm 2.75 \times 10^{-2}$
4	$2.22 \pm 2.39 \times 10^{-2}$	$2.22 \pm 2.61 \times 10^{-2}$	$2.28 \pm 3.35 \times 10^{-2}$	$2.28 \pm 2.77 \times 10^{-2}$	$2.30 \pm 2.20 \times 10^{-2}$	$2.30 \pm 3.86 \times 10^{-2}$
5	$3.72 \pm 3.45 \times 10^{-2}$	$3.72 \pm 3.09 \times 10^{-2}$	$3.86 \pm 3.77 \times 10^{-2}$	$3.86 \pm 4.62 \times 10^{-2}$	$4.59 \pm 4.29 \times 10^{-2}$	$4.59 \pm 1.80 \times 10^{-2}$
6	$3.85 \pm 2.30 \times 10^{-2}$	$3.85 \pm 2.68 \times 10^{-2}$	$4.10 \pm 2.03 \times 10^{-2}$	$4.10 \pm 2.92 \times 10^{-2}$	$4.64 \pm 1.56 \times 10^{-2}$	$4.64 \pm 1.56 \times 10^{-2}$
7	$3.86 \pm 2.12 \times 10^{-2}$	$3.86 \pm 4.26 \times 10^{-2}$	$3.99 \pm 1.90 \times 10^{-2}$	$3.99 \pm 1.67 \times 10^{-2}$	$4.72 \pm 1.80 \times 10^{-2}$	$4.72 \pm 3.24 \times 10^{-2}$
8	$3.90 \pm 2.70 \times 10^{-2}$	$3.90 \pm 3.99 \times 10^{-2}$	$4.08 \pm 2.99 \times 10^{-2}$	$4.08 \pm 2.20 \times 10^{-2}$	$4.71 \pm 2.89 \times 10^{-2}$	$4.71 \pm 2.07 \times 10^{-2}$
9	$3.83 \pm 2.49 \times 10^{-2}$	$3.83 \pm 1.61 \times 10^{-2}$	$4.06 \pm 3.02 \times 10^{-2}$	$4.06 \pm 1.89 \times 10^{-2}$	$4.77 \pm 2.20 \times 10^{-2}$	$4.77 \pm 2.41 \times 10^{-2}$
10	$3.85 \pm 3.39 \times 10^{-2}$	$3.85 \pm 3.39 \times 10^{-2}$	$4.03 \pm 1.96 \times 10^{-2}$	$4.03 \pm 1.58 \times 10^{-2}$	$4.62 \pm 2.85 \times 10^{-2}$	$4.62 \pm 2.22 \times 10^{-2}$
11	$3.74 \pm 4.34 \times 10^{-2}$	$3.69 \pm 2.20 \times 10^{-2}$	$3.79 \pm 2.36 \times 10^{-2}$	$3.78 \pm 3.05 \times 10^{-2}$	$3.59 \pm 2.37 \times 10^{-2}$	$3.77 \pm 2.20 \times 10^{-2}$
12	$3.73 {\pm} 3.24 \times 10^{-2}$	$3.25 \pm 2.50 \times 10^{-2}$	$3.81 \pm 2.45 \times 10^{-2}$	$3.38 \pm 2.29 \times 10^{-2}$	$3.68 \pm 2.16 \times 10^{-2}$	$3.38 \pm 2.28 \times 10^{-2}$
13	$3.70 {\pm} 2.20 \times 10^{-2}$	$3.30 \pm 3.35 \times 10^{-2}$	$3.79 \pm 2.57 \times 10^{-2}$	$3.25 \pm 2.74 \times 10^{-2}$	$3.63 \pm 2.03 \times 10^{-2}$	$3.25 \pm 2.70 \times 10^{-2}$
14	$3.71 \pm 2.20 \times 10^{-2}$	$3.30 \pm 2.31 \times 10^{-2}$	$3.72 \pm 2.79 \times 10^{-2}$	$3.30 \pm 2.43 \times 10^{-2}$	$3.74 \pm 3.22 \times 10^{-2}$	$3.34 \pm 2.55 \times 10^{-2}$
15	$3.73 \pm 4.00 \times 10^{-2}$	$3.30 \pm 2.50 \times 10^{-2}$	$3.83 \pm 3.07 \times 10^{-2}$	$3.38 \pm 2.29 \times 10^{-2}$	$3.60 \pm 2.13 \times 10^{-2}$	$3.35 \pm 2.37 \times 10^{-2}$
16	$3.72 \pm 3.80 \times 10^{-2}$	$3.30 \pm 2.02 \times 10^{-2}$	$3.81 \pm 2.28 \times 10^{-2}$	$3.38 \pm 2.34 \times 10^{-2}$	$3.68 \pm 3.07 \times 10^{-2}$	$3.36 \pm 2.69 \times 10^{-2}$
17	$3.73 \pm 2.50 \times 10^{-2}$	$3.32 \pm 3.63 \times 10^{-2}$	$3.79 \pm 2.51 \times 10^{-2}$	$3.40 \pm 1.82 \times 10^{-2}$	$3.68 \pm 2.22 \times 10^{-2}$	$3.38 \pm 3.05 \times 10^{-2}$

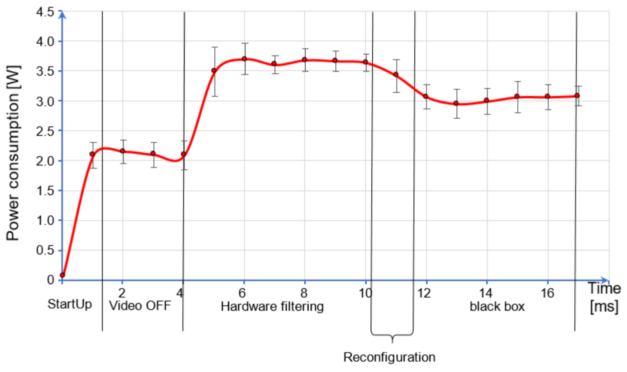


(a)

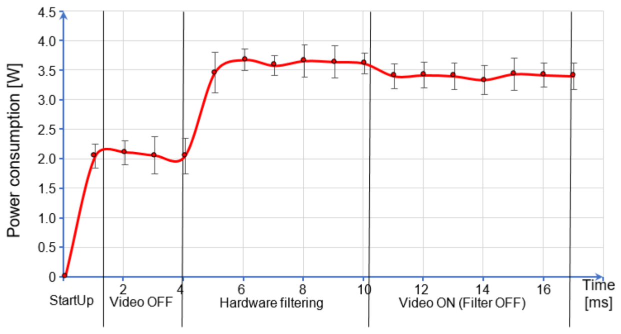


(b)

Figure 7. Power consumption versus time plot concerning the two different systems for the posterize-filtering DSP: (**a**) the dynamic partially reconfigurable architecture; (**b**) the static architecture. For each reported value, obtained as average of the 15 performed measurements, the error bar denoting the standard deviation is also shown.

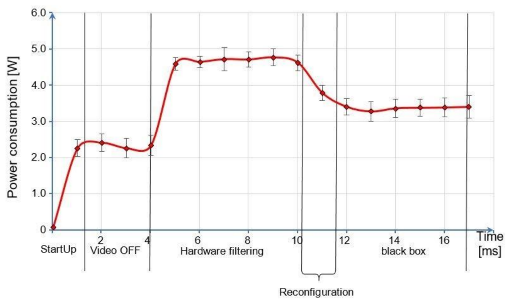


(**a**)

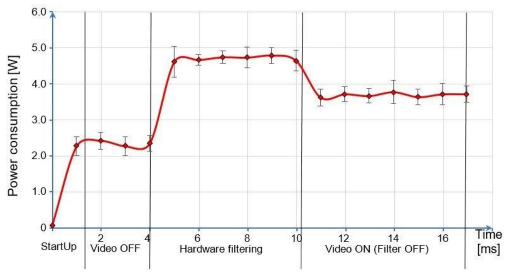


(**b**)

Figure 8. Power consumption versus time plot concerning the two different systems for the Sobel filtering DSP: (**a**) the dynamic partially reconfigurable architecture, and (**b**) the static architecture. For each reported value, obtained as average of the 15 performed measurements, the error bar denoting the standard deviation is also shown.



(**a**)



(**b**)

Figure 9. Power consumption versus time plot concerning the two different systems for the FAST filtering DSP: (**a**) the dynamic partially reconfigurable architecture, and (**b**) the static architecture. For each reported value, obtained as average of the 15 performed measurements, the error bar denoting the standard deviation is also shown.

More interestingly, these plots reveal the main differences between the partially reconfigurable architecture and the static architecture phases:

- The first three stages are the same for the two architectures: "StartUp" is the initial configuration when the device is started; then the device is in not capture video mode ("video OFF"); the operations, executed when the video is captured, and the system that performs the "hardware filtering", are the same for both architectures;
- The fourth stage is different: for the partially reconfigurable architecture there is a first period when the process of reconfiguration needs more energy (but less than the filter), and then the black box is loaded. For the static architecture, there is only one stage, because the filter is switched off.

It can be seen how the additional energy required by the reconfiguration process is later regained thanks to the black box configuration.

For completeness, Table 6 shows the distribution of the energy consumption among the principal logic components that are used by the hardware implementation of the filters. The elements generally used are Look Up Tables (LUTs), Block RAM (BRAM), DSP, and signaling components. The values were obtained by the Vivado Power Analysis tool, which can analyze the configuration of the whole design and estimates the power consumption for each module.

Table 6. The power consumption concerning the main elements of the hardware filters.

Configuration	LUTs (W)	BRAM (W)	DSP (W)	Signals (W)	Total (W)
posterize	0.035	0.025	0.009	0.047	0.116
Sobel operator	0.040	0.026	0.009	0.050	0.125
FAST	0.045	0.184	0.009	0.058	0.296

7. Discussion

The proposed case study aims to evaluate which conditions enable the energy efficiency of a dynamic, partially reconfigurable, FPGA-based device. The presented results could be used in the IoT domain, as well as in ubiquitous and edge computing, in which the most of nodes are characterized by limited resources and require high-efficiency and low-power consumption [48].

Since the 1980s, most embedded systems have used microprocessor-based kits for indoor solutions [49], which do not require a large amount of processing power. However, in the context of the IoT, the devices need more computational efficiency to appropriately handle multiple flows of information and the related computation. Most of those devices are provided with a reliable connection, a processing core with parallelism feature, a high degree of internal reconfigurability, and adequate power supply sources. Especially, when these power supply sources are batteries that cannot be easily recharged, power-saving is one of the main key aspects. In this case, the high-end FPGA devices, coupled with a high-performance embedded microprocessor, can play a significant role for the goal of energy efficiency [50]. So, the main point is how the dynamic partial reconfiguration of the latest FPGA devices may be used for this goal.

The work presented in this paper aimed at exploiting a device, which combines an FPGA and a microprocessor in the same chip, to realize a dynamic, partially reconfigurable system and use this paradigm for energy efficiency.

Unlike static systems, a dynamic, partially reconfigurable device can support different hardware modules that can be interchanged at runtime. If some of these modules do not implement logic (i.e., they contain an empty module that is also called a black box), they can be loaded instead of the processing logic so that the portion of the device consumes less energy.

Unfortunately, the configuration process itself introduces a further energy overload. So, the conditions that save energy, using a partially reconfigurable hardware accelerator, have been investigated. We compared the reconfigurable architecture with two different systems: the first one uses the same filters to process the video stream at software-level, while the second one is a non-reconfigurable architecture. For an accurate and concrete evaluation, an analytical framework of the used parameters was provided, which succeeding in generalizing the problem by means of three different aspects:

1. Quantifying the difference of power consumption between a software and a hardware system, in terms of the power consumption required to perform the same operation using either a microprocessor or an embedded hardware device. This allows one to understand under which timing conditions a dedicated hardware can provide energy-savings compared to the corresponding software implementation;
2. Evaluating if the energy needed by the management of the reconfiguration process could invalidate the energy saved by the reconfiguration process;
3. In the specific case of dynamic partial reconfiguration compared to a non-reconfigurable system, assessing if it is possible to achieve a significant benefit using this paradigm.

The estimated timing conditions that allow the dynamic partially reconfigurable process to achieve relevant energy efficiency with respect to the corresponding static architecture are variable and depend on the energy consumption of each reconfigured IP core. The paper presents a deep analysis that links the time between two dynamic partial reconfigurations and the reconfigured core energy consumption for several energy saving rates (25%, 50%, 75%, and 99.9%). Considering the posterize filter, a reconfiguration period of 663 ms enables a 99.9% energy saving rate in a partially reconfigurable device when compared to the same static device with the posterize filter always on. Similar conclusions can be derived for the other filter implementations that have different energy consumption values.

8. Conclusions

The proposed paper investigated under which conditions a partially reconfigurable hardware accelerator can provide energy efficiency for complex processing tasks by introducing a more general analytical framework for a direct comparison between the energy efficiency of a dynamic, partially reconfigurable device and a static non-reconfigurable one. Accordingly, a specific case study was implemented and analyzed on a FPGA based device. Two different solutions were compared to perform the same video processing task using three different filters, namely, posterize, Sobel operator, and FAST. In the first case, the use of software filtering allowed one to achieve CPU performance and consumption, while in the second case the same parameters were measured by loading the hardware filter module. Three main aspects were considered to determine the energy efficiency conditions:

- Comparison between the energy ratio and the processing time ratio of the hardware and software cases;
- Comparison of the energy saved through partial dynamic reconfiguration and the exceeding energy due to the reconfiguration process;
- Comparison between the energy efficiency of the reconfigurable architecture against the static architecture.

By carrying out a set of power consumption measurements, it was shown that for burdensome computational operations, such as in our video processing application, the difference in energy efficiency between software and hardware is considerable. Similar results were achieved for all the three hardware filters, with the highest energy efficiency for the posterize filter reaching a 96.6% saving factor compared to the software version. In the hardware versus software comparison, the CPU has a working frequency higher than the FPGA, but the latter is an ad-hoc specialized architecture and not a general-purpose one. This characteristic led to the low capacity of complex calculations of the microprocessor and greatly increased the latency for processing a frame.

For the second point, it was possible to derive a formula that correlated the idle time of the hardware module with remaining known parameters. The idle time was expressed as a variable to have the freedom to choose the filter deactivation time so that the following estimations are obtained. Using the data collected earlier, it was possible to estimate that the additional energy, introduced by the components that perform the partial reconfiguration, does not invalidate the possibility of energy saving.

For the third point, we introduced an original formulation for investigating the energy efficiency in the case of a partial dynamic reconfiguration system compared to the static version. By relying on FPGA technical characteristics, as well as power consumption measurements, we estimated the timing conditions that allowed the dynamic partially reconfigurable process to achieve relevant energy efficiency with respect to the corresponding static architecture.

In conclusion, under certain timing conditions, a dynamic partially reconfigurable system can achieve a considerable energy saving factor. The main application could be in those solutions that need multiple powerful processing units, such as advanced parallel DSP systems.

Future works will aim to investigate the possibility of changing the reconfigurable area size of the chip at runtime. Exploiting this improvement, the capabilities of the dynamic partial reconfiguration might be extended over the current limits, possibly giving rise to novel techniques for energy efficiency.

Acknowledgments: This research work was supported by funds from the University of Enna KORE.

Author Contributions: Vincenzo Conti and Giuseppe Dario Billeci conceived the study; Vincenzo Conti and Giuseppe Dario Billeci designed and performed the experiments; Vincenzo Conti, Giuseppe Dario Billeci, and Leonardo Rundo analyzed the data; Vincenzo Conti, Giuseppe Dario Billeci, and Leonardo Rundo wrote the manuscript; Vincenzo Conti, Carmelo Militello, and Salvatore Vitabile supervised the design and the development of the proposed study; Carmelo Militello and Salvatore Vitabile reviewed the manuscript; and all authors read and approved the final manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Conti, V.; Vitabile, S.; Agnello, L.; Sorbello, F. Fingerprint and iris based authentication in inter-cooperative emerging e-infrastructures. In *Internet of Things and Inter-Cooperative Computational Technologies for Collective Intelligence*; Springer: Berlin, Germany, 2013.
2. Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A survey. *Comput. Netw.* **2010**, *54*, 2787–2805. [CrossRef]
3. Shen, S.; Shan, Y.; Sun, L.; Sun, J.; Zou, Z.; Wang, R. Design and implementation of low-power analog-to-information conversion for environmental information perception. *Energies* **2017**, *10*, 753. [CrossRef]
4. Militello, C.; Conti, V.; Vitabile, S.; Sorbello, F. Embedded access points for trusted data and resources access in HPC systems. *J. Supercomput.* **2011**, *55*, 4–27. [CrossRef]
5. Militello, C.; Conti, V.; Vitabile, S.; Sorbello, F. An embedded iris recognizer for portable and mobile devices. *Int. J. Comput. Syst. Sci. Eng.* **2010**, *25*, 119–131.
6. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660. [CrossRef]
7. Kaxiras, S.; Martonosi, M. *Computer Architecture Techniques for Power-Efficiency*; Synthesis Lectures on Computer Architecture; Morgan and Claypool Publishers: San Rafael, CA, USA, 2008.
8. Chhabra, S.; Solihin, Y. Green secure processors: Towards power-efficient secure processor design. In *Transactions on Computational Science X*; Springer: Berlin, Germany, 2010.
9. Akan, O.B.; Isik, M.T.; Baykal, B. Wireless passive sensor networks. *IEEE Commun. Mag.* **2009**, *47*, 92–99. [CrossRef]
10. Bryce, R. *Gusher of Lies: The Dangerous Delusions of "Energy Independence"*; Public Affairs: New York, NY, USA, 2008; ISBN 978-1-58648-321-0.
11. Liu, S.; Pittman, R.N.; Forin, A. Energy reduction with run-time partial reconfiguration. In Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA), Monterey, CA, USA, 21–23 February 2010; p. 292.
12. Raghunathan, V.; Ganeriwal, V.; Srivastava, M. Emerging techniques for long lived wireless sensor networks. *IEEE Commun. Mag.* **2006**, *44*, 108–114. [CrossRef]

13. Conti, V.; Vitabile, S. Design exploration of AES accelerators on FPGAs and GPUs. *J. Telecommun. Inf. Technol.* **2017**, *1*, 28–38.
14. Wang, S.; Liu, D.; Zhou, J.; Zhang, B.; Peng, Y. A run-time dynamic reconfigurable computing system for lithium-ion battery prognosis. *Energies* **2016**, *9*, 572. [CrossRef]
15. Chi, Q.; Yan, H.; Zhang, C.; Pang, Z.; Xu, L.D. A reconfigurable smart sensor interface for industrial WSN in IoT environment. *IEEE Trans. Ind. Inform.* **2014**, *10*, 1417–1425. [CrossRef]
16. Conti, V.; Militello, C.; Sorbello, F.; Vitabile, S. Biometric sensors rapid prototyping on field-programmable gate arrays. *Knowl. Eng. Rev.* **2015**, *30*, 201–219. [CrossRef]
17. Cardoso, J.M.P.; Hübner, M. *Reconfigurable Computing: From FPGAs to Hardware/Software Codesign*; Springer: New York, NY, USA, 2011; ISBN 978-1-4614-0060-8.
18. Hsiung, P.A.; Santambrogio, M.D.; Huang, C.H. *Reconfigurable System Design and Verification*, 1st ed.; CRC Press Inc.: Boca Raton, FL, USA, 2009; ISBN 1420062662.
19. Frangieh, T.; Chandrasekharan, A.; Rajagopalan, S.; Iskander, Y.; Craven, S.; Patterson, C. PATIS: Using partial configuration to improve static FPGA design productivity. Proceedings of IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum (IPDPSW), Atlanta, GA, USA, 19–23 April 2010; pp. 1–8.
20. Wisniewski, R. *Synthesis of Compositional Microprogram Control Units for Programmable Devices*; University of Zielona Gòra Press: Zielona Gòra, Poland, 2009; ISBN 978-83-7481-293-1.
21. Koch, D. *Partial Reconfiguration on FPGAs: Architectures, Tools and Applications*, 1st ed.; Springer: New York, NY, USA, 2013; ISBN 978-1-4899-9356-4.
22. Xilinx Inc. *Vivado Design Suite User Guide: Partial Reconfiguration (UG909)*; Xilinx Inc.: San Jose, CA, USA, 2017. Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug909-vivado-partial-reconfiguration.pdf (accessed on 18 January 2018).
23. Lie, W.; Wu, F.-Y. Dynamic partial reconfiguration in FPGAs. In Proceedings of the 3rd International Symposium on Intelligent Information Technology Application (IITA), Nanchang, China, 21–22 November 2009; Volume 2, pp. 445–448.
24. McDonald, E.J. Runtime FPGA partial reconfiguration. In Proceedings of the 2008 Aerospace Conference, Big Sky, MT, USA, 1–8 March 2008; pp. 1–7.
25. Hassan, A.; Ahmed, R.; Mostafa, H.; Fahmy, H.A.; Hussien, A. Performance evaluation of dynamic partial reconfiguration techniques for software defined radio implementation on FPGA. In Proceedings of the 2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS), Cairo, Egypt, 6–9 December 2015; pp. 183–186.
26. Pezzarossa, L.; Schoeberl, M.; Sparsø, J. A Controller for Dynamic Partial Reconfiguration in FPGA-Based Real-Time Systems. In Proceedings of the 20th IEEE International Symposium on Real-Time Distributed Computing (ISORC), Toronto, ON, Canada, 16–18 May 2017; pp. 92–100.
27. Telikepalli, A. Power vs. performance: The 90nm inflection. In *Xilinx White Paper*; Xilinx Inc.: San Jose, CA, USA, 2006.
28. Tuan, T.; Lai, B. Leakage power analysis of a 90 nm. In Proceedings of the 2003 IEEE Custom Integrated Circuits, San Jose, CA, USA, 24 September 2003; pp. 57–60.
29. Anderson, J.H.; Najm, F.N. Active leakage power optimization for FPGAs. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2006**, *25*, 423–437. [CrossRef]
30. Xilinx Inc. *Vivado Design Suite User Guide: Getting Started (UG910)*; Xilinx Inc.: San Jose, CA, USA, 2014.
31. Xilinx Inc. *Vivado Design Suite User Guide: Design Flows Overview (UG892)*; Xilinx Inc.: San Jose, CA, USA, 2017.
32. Xilinx Inc. *Zynq-7000 All Programmable SoC*; Xilinx Inc.: San Jose, CA, USA, 2018.
33. Xilinx Inc. *Zynq-7000 All Programmable SoC Data Sheet: Overview (DS190)*; Xilinx Inc.: San Jose, CA, USA, 2017.
34. Xilinx Inc. *Zynq-7000 All Programmable SoC: Technical Reference Manual (UG585)*; Cortex-A9 Processors; Xilinx Inc.: San Jose, CA, USA, 2017.
35. Avnet Inc. *ZedBoard Getting Started Guide (Version 7.0)*; Xilinx Inc.: San Jose, CA, USA, 2014.
36. Xilinx Inc. *Zynq-7000 All Programmable SoC*; Xilinx Inc.: San Jose, CA, USA, 2016.
37. Kohn, C. Partial reconfiguration of a hardware accelerator with Vivado Design Suite for Zynq-7000 AP SoC processor. In *Xilinx Application Note*; Xilinx Inc.: San Jose, CA, USA, 2006.

38. Greenberg, I.; Xu, D.; Kumar, D. *Processing: Creative Coding and Generative Art in Processing 2*; Apress Media LLC: New York, NY, USA, 2012; ISBN 143024464X.
39. Sobel, I.; Feldman, G. A 3 × 3 isotropic gradient operator for image processing. In *Pattern Classification and Scene Analysis*; Duda, R., Hart, P., Eds.; John Wiley & Sons: New York, NY, USA, 1968; pp. 271–272.
40. Rosten, E.; Drummond, T. Fusing points and lines for high performance tracking. In Proceedings of the 10th IEEE International Conference on Computer Vision (ICCV), Beijing, China, 17–21 October 2005; Volume 2, pp. 1508–1515.
41. Xilinx Inc. *XAPP1231—Partial Reconfiguration of a Hardware Accelerator with Vivado Design Suite*; Xilinx Inc.: San Jose, CA, USA, 2014.
42. Cuoccio, A.; Grassi, P.R.; Rana, V.; Santambrogio, M.D.; Sciuto, D. A generation flow for self-reconfiguration controllers customization. In Proceedings of the 4th IEEE International Symposium on Electronic Design, Test and Applications (DELTA), Hong Kong, China, 23–25 January 2008; pp. 279–284.
43. Claus, C.; Zhang, B.; Stechele, W.; Braun, L.; Hübner, M.; Becker, J. A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput. In Proceedings of the 2008 International Conference on Field Programmable Logic and Applications (FPL), Heidelberg, Germany, 8–10 September 2008; pp. 535–538.
44. Papadimitriou, K.; Dollas, A.; Hauck, S. Performance of partial reconfiguration in FPGA systems: A survey and a cost model. *ACM Trans. Reconfig. Technol. Syst.* **2011**, *4*, 36. [CrossRef]
45. Al Kadi, M.; Rudolph, P.; Gohringer, D.; Hübner, M. Dynamic and partial reconfiguration of Zynq 7000 under Linux. In Proceedings of the 2013 IEEE International Conference on Reconfigurable Computing and FPGAs (ReConFig), Cancun, Mexico, 9–11 December 2013; pp. 1–5.
46. Bonamy, R.; Bilavarn, S.; Chillet, D.; Sentieys, O. Power consumption models for the use of dynamic and partial reconfiguration. *J. Microprocess. Microsyst.* **2014**, *38*, 860–872. [CrossRef]
47. Paulsson, K.; Hübner, M.; Bayar, S.; Becker, J. Exploitation of run-time partial reconfiguration for dynamic power management in Xilinx Spartan III-based systems. In Proceedings of the International Workshop on Reconfigurable Communication-Centric System-on-Chip (ReCoSoC), Montpellier, France, 18–20 June 2008.
48. Conti, V.; Rundo, L.; Militello, C.; Mauri, G.; Vitabile, S. Resource-efficient hardware implementation of a neural-based node for automatic fingerprint classification. *J. Wirel. Mob. Netw. Ubiquit. Comput. Depend. Appl.* **2017**, *8*, 19–36. [CrossRef]
49. Gilchrist, A. *IOT Security Issues*; Walter De Gruyter GmbH: Berlin, Germany, 2017; ISBN 978-1-5015-0577-5.
50. Keramidas, G.; Voros, N.; Hübner, M. *Components and Services for IoT Platforms: Paving the Way for IoT Standards*, 1st ed.; Springer International Publishing: Cham, Switzerland, 2017.