Low Power Image Processing Applications on FPGAs using Dynamic Voltage Scaling and Partial Reconfiguration

Ariel Podlubne, Julian Haase, Lester Kalms, Gökhan Akgün,
Muhammad Ali, Habib ul hasan Khan, Ahmed Kamal and Diana Göhringer
Technische Universität Dresden, Germany
{ariel.podlubne, julian.haase, lester.kalms, goekhan.akguen,
muhammad.ali, habib.khan, ahmed.kamal, diana.goehringer}@tu-dresden.de

***Abstract*—The TULIPP project aims to facilitate the development of embedded image processing systems with real-time and low-power constraints. In this paper, several adaptive dynamic runtime techniques for reconfigurable SoCs are described. These methods are used for low power image processing applications on high-performance embedded platforms. Dynamic voltage scaling and dynamic partial reconfiguration target the low-power requirements of the embedded systems while debugging supports the fast development on the hardware side of the system. The proposed techniques were tested and verified using an own developed custom SDSoC image processing library.**

***Index Terms*—Embedded systems, image processing, real-time, reconfigurable, low power, FPGA, Dynamic Voltage Scaling, Dynamic Partial Reconfiguration , Debugging**

I. Introduction and Motivation

Image processing and their applications are a wide and complex domain due to e.g. large data volumes, high performance requirements and extensive sensor types. Moreover, image processing itself often aims at case scenarios where power, energy consumption, weight, physical size and costs are important during the development phase. Designers phase these restrictions and the challenges they represent have to be resolved. The comprehensive purpose of the *Towards Ubiquitous Low-power Image Processing Platforms* (TULIPP) project is to lower the factor of the given challenges by providing a complete image processing reference platform through associated guidelines so that developers are able to achieve their goals in embedded image processing applications [1].

We propose two runtime techniques to achieve the low power consumption requirement. On one hand, the Dynamic Voltage Scaling (DVS) technique can modify the voltage during runtime depending on the needed performance of the application. In this paper, we discuss a software solution for DVS and power monitoring on ZC702 development board by running FreeRTOS and using Xilinx SDSoC. On the other hand, Dynamic Partial Reconfiguration (DPR) modifies only a defined region of the hardware without affecting the static part. In this work, the EMC^2-TULIPP hardware instance based on Xilinx's Zynq-7000 System-on-Chip (SoC) is used for the DPR technique. This method reduces the amount of hardware utilization of the Field Programmable Gate Array (FPGA) and as a consequence the static power consumption.

In this work, we used FreeRTOS to make use of the proposed techniques to achieve real-time constraints in the software of the Processing System (PS). Moreover, the debugging system software driver is developed for FreeRTOS so that a fast hardware validation during runtime is possible.

The paper is organized as follows: Sections II presents related work and Section III provides background information. Section IV discusses scheduling techniques to achieve low power consumption. Section V shows the DPR technique and debugging support is introduced in Section VI. Section VII presents the results and the paper is concluded in Section VIII.

II. Related Work

Dynamic voltage scaling involves changing the voltage of the circuit in order to reduce power consumption. Lowering the voltage of the circuit results in a decrease of the dynamic and leakage current. Although, it also increases the gate delay. The idea is to scale voltage as low as possible and maintain a reliable performance of the design [2]. Many different implementations have already been done in this field to improve the power consumption of FPGAs. A software and hardware method for DVS is described in [3]. A complete PL-side setup, using MicroBlaze IP, Dual-Port RAM (DPRAM) and I^{2}C was defined in order to scale the voltage and perform power monitoring. In [4], the importance of Logic Delay Measurement Circuit (LDMC) is described. First, voltage scaling is done and then a suitable frequency is selected on which the system should operate. Consequently, gate delay can be adjusted. Ishihara et al. [5] present a self-adaptive voltage control in order to solve the problems of implementing Dynamic Voltage and Frequency Scaling (DVFS) for low power FPGAs. In this paper, we describe a software solution to implement dynamic voltage scaling and power measurement using FreeRTOS and Xilinx SDSoC on ZC702.

Current FPGAs like the Xilinx Zynq-7000 SoC offer the possibility to reconfigure some defined area of the hardware dynamically which is called DPR [6]. Several papers discussed the benefits of using this technique for image processing

978-1-5386-8237-1/18/$31.00

in real-time reconfigurable designs [7]–[10]. DPR is used to switch between different image processing applications during run-time by sharing the same hardware resources on the FPGA. Therefore, the design can fit on a smaller FPGA, which results in lower power consumption and costs. A run-time reconfigurable MPSoC architecture for future driver assistance systems has been developed by [9]. Their proposed architecture is based on run-time reconfigurable hardware accelerator engines for video processing targeting a Xilinx Virtex-2 FPGA. In [10] a fast configuration engine is designed and deployed in the system. The engine shows that it can outperform the engine provided with Zynq SoC by three times. This is due to the fact that the configuration engine for Partial Reconfiguration (PR) is completely implemented in the Programmable Logic (PL)-side. Nunez-Yanez et al. [11] propose a power adaptive architecture that includes DVFS, independent from the PS-side. Moreover, the architecture supports DPR via the PS-side. In our work, these two techniques are controlled from the PS-side where an operating system is running and image processing application cores are implemented as case studies.

III. Background

This section gives background information to TULIPP, FreeRTOS and the images processing library, since they are used in this work.

A. TULIPP

The main of objective of TULIPP [1] is to offer a reference platform. The aim is a fast development of embedded image processing applications. Several design and implementation rules as well as interfaces between the components of the platform are provided, so that developers are able to design a matching platform. The reference platform shown in Figure 1 is composed of three layers: hardware architecture, real time operating system including libraries and tool chain. The hardware architecture is a template of the heterogeneous computing system like the Xilinx Zynq architecture. The real time operating system and low level libraries are designed to meet image processing application requirements and run on the instantiated of the hardware layer. The tool chain captures the given application and maps it to the heterogeneous hardware.

B. FreeRTOS

FreeRTOS is a portable and open-source operating system suited for embedded applications. It ensures real-time requirements of an application at run-time. FreeRTOS supports semaphores and mutex to share and synchronize resources. A real-time scheduler allows to execute different functions in independent threads. It also enables the execution of a thread depending on its priority. The designer can assign a thread with a higher priority due to hard real-time requirements [12]. Furthermore, three type of scheduling algorithms can be chosen from the FreeRTOS kernel. In this paper, the fixed-priority preemptive scheduling algorithm is running with an image processing application that executes its threads with the same priority by using the Round-Robin policy [13].

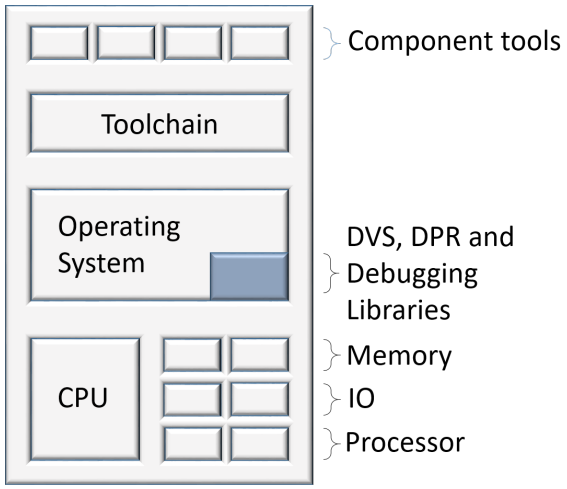

Fig. 1. The TULIPP reference platform

C. Image Processing Library

This work uses an image processing library to test and verify the proposed techniques. This library has been developed and implemented in the TULIPP project [14]. We have implemented it in C++ for SDSoC using High Level Synthesis (HLS). It is a template-based library, which provides basic image processing functions for streaming applications. The library supports various data types and auto-vectorization for different functions. The main advantages of this library are that it is independent from the SDSoC version, it does not require extra libraries like OpenCV and it is not dependent on an operating system. It supports more than 21 function of the OpenVX specification, such as Gaussian, Median or Sepia filter. All of them take an input-stream and convert them into an output-stream using the AXI-Stream interface. With a common interface, different functions can be easily replaced by using partial reconfiguration.

IV. Dynamic Voltage Scaling Technique with FreeRTOS for an Image Processing Application

Dynamic voltage scaling is a power management technique that controls the voltage based on the workload, fabrication and operating condition of a system [3]. During the development stage of an application, it is required to optimize the power dissipation of the hardware and improve the performance of the HW/SW-Codesign. This dissipation can be changed through appropriate DVS-technique at runtime. Xilinx Zynq-7000 SoC ZC702 (EK-Z7-ZC702-G) provides suitable hardware components to measure and control the power dynamically. The SoC communicates with the power controller through the Power Management Bus (PMBus) using the I^2C interface. The core and auxiliary voltages are supplied to the board using power regulators. An application ensures from the PS-side that the desired power is achieved. Furthermore, it monitors dynamically the power dissipation of the reconfigurable hardware.

TABLE I
OVERVIEW OF THE SCHEDULED TASKS IN FREERTOS ON ONE ARM-CORE.

Application Function	Execution Time [ms]
Power Monitoring	262.05
Image Processing Application	95.32

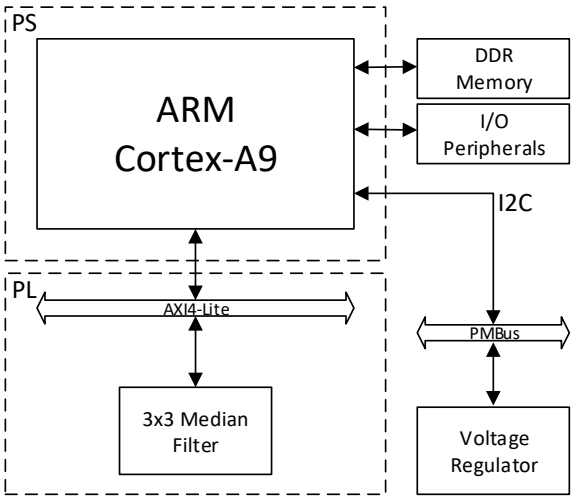

Fig. 2. System Overview. Power app: application for DVS and power measurement, Img app: Image processing application with hardware accelerated function on PL

The application is built on two main tasks. The first one monitors the power dissipation and controls dynamically the voltage scaling of the PL-side. The second one contains the image processing application (as described in section III). Xilinx SDSoC permits only to use one of the dual-core processors so that an operating system was required to schedule both functions. FreeRTOS is also deployed to keep the soft real-time requirement of the application. A preemptive priority-based scheduler, guarantees that either the power monitoring task or the image processing application runs at each clock cycle. The power monitoring (described as Task 1) has a higher priority than the image processing application (described as Task 2). At the beginning, Task 1 starts to measure power of PL. Then it is blocked by a delay of approximately $95\,\mathrm{ms}$. The scheduler enables the execution of the Task 2 in that delay. Both tasks are running periodically and are executed completely for approximately $360\,\mathrm{ms}$. Task 2 is preempted multiple times by Task 1 to provide power measurement from the PL-side. Table I shows the overview of the task scheduling.

A median filter is running on the PL-side in Task 2. Xilinx SDSoC allows to accelerate functions of the software application into the PL-side of SoC. An overview of the implemented hardware architecture is shown in Figure 2.

For this project, the most interesting part is acceleration of the image processing application in hardware. Only voltage scaling and power measurement of the PL was done. This improved the total measurement time of power consumption by 14 times as compared to measuring all voltage rails, due to less computation. The voltage rails of the board allow to measure voltage and current of each rail supplied to the board. Voltage scaling of PL was adapted by setting voltage level in software and can be set dynamically.

V. DYNAMIC PARTIAL RECONFIGURATION

DPR offers the flexibility to reconfigure a design with different Reconfigurable Modules (RMs) at runtime reusing the same hardware resources on the FPGA floorplan. A DPR design consists of a static part and one or more Reconfigurable Partitions (RPs) that are dynamically reconfigured with the desired Reconfigurable Module (RM), depending on the design. A set of partial bitstreams are generated for all of the RMs of the reconfigurable system. Xilinx's DPR technique is used on this paper by dynamically reconfiguring the RPs through the Processor Configuration Access Port (PCAP).

The PL can be configured and reconfigured by the software running on the PS using the PCAP path in the Device Configuration Interface (DevC) [15]. The Advanced eXtensible Interface (AXI)-PCAP bridge is the main component in the configuration of the PL for deployment of bitstreams. The PCAP bridge converts 32-bit AXI formatted data into the 32-bit PCAP protocol and vice versa [16].

The DMA-Controller (DMA) included in the PCAP transfers the data of the bitstream between the memory device, usually the Double Data Rate (DDR) memory, into the First In First Out (FIFO) registers of the PCAP bridge over an AXI-Interconnect. The DevC driver functions have to set up the PCAP bridge into the correct mode and initialize the DMA transfer to send the full and the partial bitstream from the memory device to the FIFOs. In non-secure mode, the transfer rate through the PCAP bridge is around $145\,\mathrm{MB/s}$. The PL configuration module can accept data at the rate of 32 bits per PCAP clock, but the overall throughput is limited by the PS AXI interconnect [15].

The design shown in Figure 3 includes an AXI-DMA, a FIFO and two HLS cores, which will act as RMs. It has been implemented on the TULIPP hardware instance, which is based on the EMC^2-Z7030 board. Two case studies are presented here. One core is a passthrough and the second one is a sepia filter. They have been designed to have the same entity to apply the DPR technique. Therefore, they are composed by an input and an output AXI-Stream interface for streaming the images, as described in Section III-C.

At the beginning, one RM is reconfigured. Then, an image is read from a Secure Digital (SD) card, loaded into memory and streamed via the AXI-DMA to the filter. Its output is read back from the AXI-DMA to later save the result into the SD card. It is important to highlight that any filter, based on the image processing library described in Section III-C, can be included in this design to be partially reconfigured.

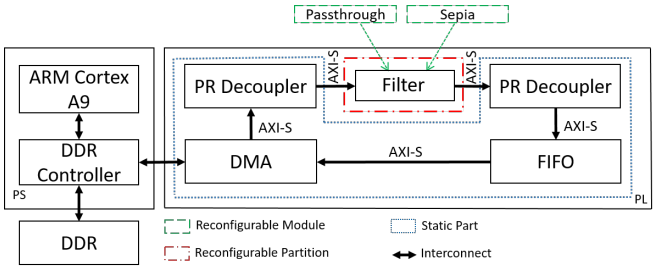

Fig. 3. Static design for the case studies.

VI. Debugging Support

During the design phase, many bugs are expected. If they are encountered during the simulation phase, they are easy to solve. However, in many cases, the bugs are encountered during the hardware phase. Hardware validation and verification are the most critical steps because of hardware invisibility. Even in case of Integrated Logic Analyzers (ILAs), the monitored signals are visible only for few clock cycles depending upon the hardware resources. We presented a debugging system which can provide complete visibility and lossless stream of data, effectively with an unlimited debug window [17]. The processor-based debugging system is utilized to collect the data from onboard trace buffers. Once they are full, the Device Under Test (DUT) is stopped by the clock manager and then the data is transferred to the terminal for processing through the available communication port. After the data transfer, the debugging system starts clocking the DUT again and the iterative process can continue. The data can be viewed by any waveform viewer. The block diagram of the hardware platform is shown in Figure 4.

The debugging system comprises of three modules namely the hardware platform, the debugging system software driver and a Graphical User Interface (GUI) running on the terminal. The software driver has been developed for FreeRTOS using C/C++ programming language. However, the code is generic and can be ported to any other operating system with minimal effort. The block diagram of the software is shown in Figure 5.

The debugging system driver is based upon the drivers of different peripherals used in the debugging system. The debugging system contains a DMA, interrupt controller and GPIOs. Based upon the same analogy, the main ingredients of the driver are interrupt setup, DMA initialization, debugging system re-initialization in the interrupt handler and the data transmission through serial data communication. In order to setup the debugging system, we used the same base addresses included in the specification files. The debugging system also needs to communicate with the terminal in order to transfer data. Serial data communication has been used so that other available communication channels like Ethernet, PCI etc. can be used for other application. At the terminal side, the GUI is used to setup the serial port. Then it starts receiving the data. Once the data has been received and no more data transfer is required, the communication can be stopped through the GUI. The debugging data is then plotted. After the completion of the process, the GUI can be used to close the serial port and save the data which can be post-processed afterwards.

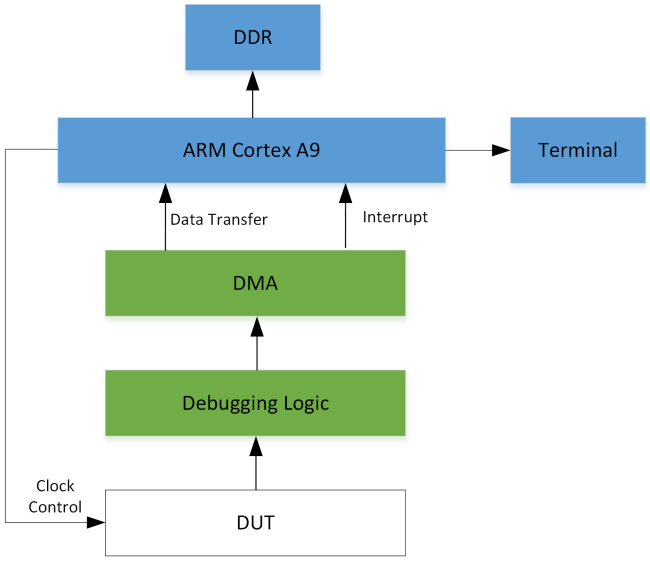

Fig. 4. Hardware platform for debugging system

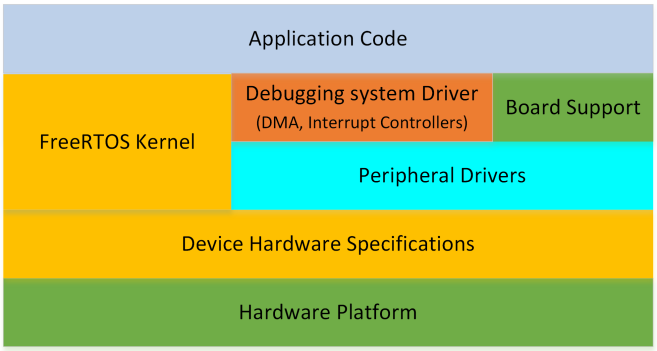

Fig. 5. Debugging system software

VII. Evaluation

A. Dynamic Voltage Scaling

DVS was tested by scaling the voltage to different levels dynamically and monitoring the power consumption of the PL. The image processing function used for evaluation of DVS is a 3x3 Median Filter for 1920x1080 16-bit images. It is processing 1-pixel per clock cycle (plus overhead). The resource utilization of the entire system with the filter median moved to the PL-side (operating at 100 MHz) is shown in Table II.

The voltage of normal execution of the image processing application without any voltage scaling was approximately 1.0V and was taken as reference. For evaluation, different voltage levels were set at 0.85V, 0.90V and 0.95V. Average power consumption after voltage scaling was measured and

TABLE II
RESOURCE UTILIZATION OF THE MEDIAN FILTER

Resource	Used	Total	% Utilization
DSP	0	220	0
BRAM	18	140	12.86
LUT	9842	53200	18.5
FF	11984	105400	11.26

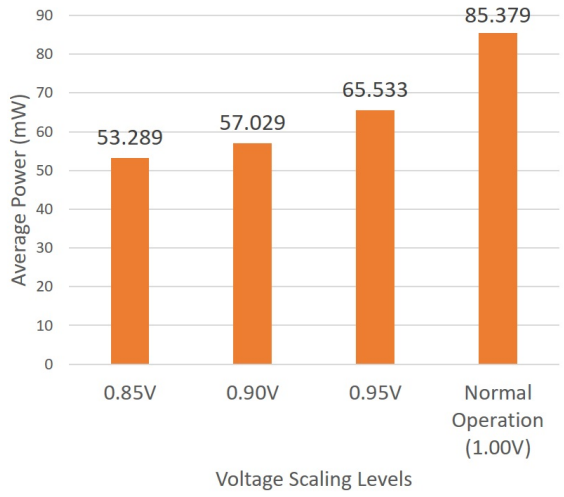

Fig. 6. Average power consumption

compared with power consumption of the normal operation. Figure 6 shows the results of the test.

It can be noted from the chart that power consumption with scaling is lower as compared with normal execution of image processing application. For 0.95V scaling, power consumption is reduced by 23.2%, for 0.90 V scaling, it is reduced by 33.2% and for 0.85V scaling, it is reduced by 37.6% as compared to normal execution without any scaling.

It was also interesting to note the execution time of the median filter. With hardware acceleration, the median filter executed at an average time of 20.880 ms and execution on software took an average of 653.785 ms. This means that with hardware acceleration the median filter executed 31 times faster as compared to software execution.

DVS results show that power consumption can be reduced up to 37% using voltage scaling. This can be improved if dynamic frequency scaling is also added to this project. For frequency scaling, a custom SDSoC platform would be needed for implementation. A performance parameter will be helpful in implementing a complete DVFS system. It will help to improve both power consumption and performance of the system.

B. Dynamic Partial Reconfiguration

DPR is running as a standalone application. As described in Section V, two RM can be changed at runtime so that either the passthrough or sepia filter runs after the reconfiguration. The images used for the case studies are 256x256 RGB in 32 bits. Table III shows the execution time of each function.

TABLE III
OVERVIEW ABOUT THE RUNNING SOFTWARE TASK WITH DYNAMIC PARTIAL RECONFIGURATION

Application Function	Execution Time [ms]
Bitstream Loading	517.050
Initialization	0.034
Reconfiguration Time	10.000
Reconfigurable Module	1.316

At the beginning, all available partial bitstreams (1,250 Kbytes each) are loaded from the SD card into the memory. Then, the PCAP-controller is initialized to reconfigure only one core at runtime. Despite having two cores, their reconfigurable times are the same due to the fact that all partial bitstream have the same size.

C. Debugging support

In order to test the debugging system, a Gaussian filter was used as DUT. The filter has a window generator in which the image width, height and size of input pixel data can be specified. For the current research work, input image comprises of 1000 x 1700 pixels with each pixel represented by 8 bits. The second stage of the filter is a Gaussian 7x7 kernel. Output of Gaussian kernel is a 16-bit image pixel $(Image\_pixel\_out)$. Both of these modules are designed in VHDL.

1) Simulation Results: During the debugging phase, when the data is transferred to the terminal, any open source waveform viewer can be used for data viewing. GTKWave was used to plot the data. We monitored the input and outputs pins of the Gaussian filter without data loss and capture window limitation. In contrast, the traditional debugging approaches are limited to the capture window. For traditional approaches, an increase in capture window requires more hardware resources. Furthermore, increase in capture window is not possible beyond a certain limit. On the contrary, the debugging system used in this research work requires minimal resources and provides unlimited capture window. Hence, the presented debugging system makes debugging easier because of the unlimited window size and no data loss.

2) Resource Utilization: The debugging system can be synthesized with a range of tracing windows. Resource utilization of the presented debugging methodology for TULIPP board is synthesized with a trace window of 64 and 1024 as shown in Figure 7. 16 signals are monitored with a maximum data width of 32 bits. Resource utilization of the presented debugging system increases with an increase in tracing window, because more BRAM blocks are required as trace buffers. Debugging with a larger window is better since it permits to capture more data in one iteration and hence it is faster. However, the lossless nature of the debugging ensures the completeness of data. In case of resource-scarce designs, the 64 trace window can be used without any loss of data.

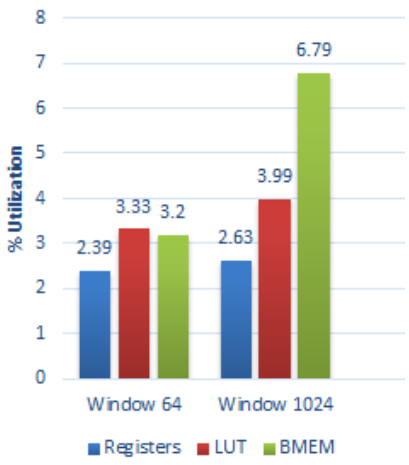

Fig. 7. Resource utilization for debugging support on a Zynq

VIII. Conclusion and Future Work

It is possible to apply techniques such as DVS and DPR, reduce the power consumption of image processing applications. As shown in Section VII, the power consumption was reduced up to 37.6% with DVS. For DPR we cam swap between different reconfigurable filters at runtime without the need to re-synthesize the entire design. Due to the high complexity of combining these approaches, the debugging supports the fast development and validation of the hardware for image processing systems. Besides, increasing its sample rate will not represent an additional overhead for the resource utilization.

One step for future work will consist of creating a custom IP, which runs on PL-side and is responsible for DVFS. This will allow it to run in parallel with any other image processing application and will give more accurate results. Later on, integrating DVFS and PR techniques together would be the next step. Besides, the concepts shown in this work will be applied for multi-cores real-time operating systems, such as HIPPEROS [18].

Acknowledgment

The work described in this paper has been supported in part by European Union's Horizon 2020 research and innovation programme project *Towards Ubiquitous Low-power Image Processing Platforms (TULIPP)* under grant agreement number 688403 and in part by the German Federal Ministry of Education and Research BMBF under grant agreement number 16KIS0666 SysKit_HW.

References

[1] T. Kalb, L. Kalms, D. Göhringer, C. Pons, F. Marty, A. Muddukrishna, M. Jahre, P. G. Kjeldsberg, B. Ruf, T. Schuchert, I. Tchouchenkov, C. Ehrenstrahle, F. Christensen, A. Paolillo, C. Lemer, G. Bernard, F. Duhem, and P. Millet, "Tulipp: Towards ubiquitous low-power image processing platforms," in *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, pp. 306–311, July 2016.

[2] C. T. Chow, L. S. M. Tsui, P. H. W. Leong, W. Luk, and S. J. E. Wilton, "Dynamic voltage scaling for commercial fpgas," in *Proceedings of International Conference on Field-Programmable Technology (FPT)*, pp. 173–180, Dec 2005.

[3] A. F. Beldachi and J. L. Nunez-Yanez, "Accurate power control and monitoring in zynq boards," in *Proceedings of 24th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, Sept 2014.

[4] A. Nabina and J. L. Nunez-Yanez, "Adaptive voltage scaling in a dynamically reconfigurable fpga-based platform," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 5, no. 4, p. 20, 2012.

[5] S. Ishihara, Z. Xia, M. Hariyama, and M. Kameyama, "Architecture of a low-power fpga based on self-adaptive voltage control," in *2009 International SoC Design Conference (ISOCC)*, pp. 274–277, Nov 2009.

[6] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford, "Invited paper: Enhanced architectures, design methodologies and cad tools for dynamic reconfiguration of xilinx fpgas," in *2006 International Conference on Field Programmable Logic and Applications*, pp. 1–6, Aug 2006.

[7] Y. Jiang and M. S. Pattichis, "A dynamically reconfigurable architecture system for time-varying image constraints (drastic) for motion jpeg," *Journal of Real-Time Image Processing*, vol. 14, no. 2, pp. 395–411, 2018.

[8] S. D. Carlo, G. Gambardella, P. Prinetto, D. Rolfo, and P. Trotta, "Safemip: A self-adaptive features extractor and matcher ip-core based on partially reconfigurable fpgas for space applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, pp. 2198–2208, Oct 2015.

[9] C. Claus, W. Stechele, and A. Herkersdorf, "Autovision–a run-time reconfigurable mpsoc architecture for future driver assistance systems (autovision–eine zur laufzeit rekonfigurierbare mpsoc architektur für zukünftige fahrerassistenzsysteme)," *it-Information Technology*, vol. 49, no. 3, pp. 181–187, 2007.

[10] J. Khalifat, A. Ebrahim, A. Adetomi, and T. Arslan, "A dynamic partial reconfiguration design for camera systems," in *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pp. 1–7, June 2015.

[11] J. L. Nunez-Yanez, M. Hosseinabady, and A. Beldachi, "Energy optimization in commercial fpgas with voltage, frequency and logic scaling," *IEEE Transactions on Computers*, vol. 65, pp. 1484–1493, May 2016.

[12] R. Berry, *Mastering the FreeRTOS Real Time Kernel*. Real Time Engineers Ltd., 2016.

[13] R. Inam, J. Mäki-Turja, M. Sjödin, S. M. Ashjaei, and S. Afshar, "Support for hierarchical scheduling in freertos," in *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pp. 1–10, IEEE, 2011.

[14] A. Sadek, A. Muddukrishna, L. Kalms, A. Djupdal, A. Podlubne, A. Paolillo, D. Göhringer, and M. Jahre, "Supporting utilities for heterogeneous embedded image processing platforms (sthem): An overview," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, (Cham), pp. 737–749, Springer International Publishing, 2018.

[15] "Zynq-7000 SoC Technical Reference Manual UG585 (V1.12.2)." Technical Reference Manual, Zynq-7000 All Programmable SoC, Xilinx, Inc., July 2018.

[16] M. A. Kadi, P. Rudolph, D. Göhringer, and M. Hübner, "Dynamic and partial reconfiguration of zynq 7000 under linux," in *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pp. 1–5, Dec 2013.

[17] H. ul Hasan Khan and D. Göhringer, "Fpga debugging by a device start and stop approach," in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1–6, Nov 2016.

[18] A. Paolillo, O. Desenfans, V. Svoboda, J. Goossens, and B. Rodriguez, "A new configurable and parallel embedded real-time micro-kernel for multi-core platforms," *OSPERT 2015*, p. 25, 2015.