Evaluating Heterogeneous Architectures based on Zynq AP SOC for Real-Time Video Processing

Fanny Spagnolo, Stefania Perri, Pasquale Corsonello
Department of Electronics, Computer Sciences and Systems
DIMES - University of Calabria
Arcavacata di Rende, Italy
e-mail: f.spagnolo@dimes.unical.it, perri@dimes.unical.it, p.corsonello@unical.it

Abstract—**Embedded systems are known as valid candidates to efficiently support image and video processing algorithms. Their high flexibility, speed performances and low power consumption are mainly due to the joint design of their software and specific hardware portions. The Zynq All Programmable System on Chip, that integrates ARM processor and high performance programmable logic resources, is nowadays often preferred to the more traditional realization platforms, such as Application Specific Integrated Circuits (ASICs) and Digital Signal Processors (DSPs). In this paper, we evaluate two different design strategies, each with its own Zynq-based support platform, giving to the designers useful hints on how to identify the best design choices for the target application. The first support platform presented here also makes use of an embedded operating system (OS); it significantly limits the required design efforts and time-to-market. The second architecture is realized without the OS support, and of course reaches much higher performances than the former, but requiring higher development and verification times. Both platforms exploit a hardware accelerator for the function of interest. As a case study, a simple but complete image processing architecture has been designed by using both the above platforms. Performances measurements revealed that an approximate speed improvement between 4 and 52 times could be obtained with respect to an all-software implementation.**

Keywords–Embedded Systems; Image Processing; Zynq .

I. INTRODUCTION

In the last few years, the development of even more complex and computationally intensive video processing algorithms has been made possible due to the ever-increasing technology progress. Many of these algorithms are adopted in a large variety of applications where real-time performances play an important role. In these cases, software-oriented implementations, running on general purpose CPUs, might not satisfy the tight speed constraints. Faster and more efficient implementations can be achieved with the aid of hardware accelerators that allow exploiting proper computational parallelisms. Embedded systems are a well known approach to speed up image and video processing algorithms by conjunct software/hardware special designs [1]. Such heterogeneous architectures allow trading off the advantages offered by the flexible software and the high performance hardware portions of the design [2]. Nowadays, among several possible realization platforms, the FPGA-based is the most interesting one. Its reduced design efforts and time-to-market make such an approach more appreciated than those based on ASICs fabrication [3]. The lower power dissipation and higher speed performances attainable by using such realization strategy make it preferable with respect to the DSP-based counterparts [4].

Usually, designing an embedded system for video processing, the designer must take into account that most algorithms perform both pixel-level and frame-level computations. Due to their higher computational complexity, pixel-level processing have a great benefit by the inclusion in the embedded system of a dedicated hardware accelerator. On the contrary, frame-level elaborations often process only few frame descriptors. Thus, they do not represent a bottleneck for the overall application. In this case, a pure software implementation can be easier, more flexible and does not compromise the system performances. It is then clear why, in order to conjugate the high-speed capability of a hardware implementation with the flexibility provided by a software design, heterogeneous System-on-Chips (SoCs) based on FPGA have been recently recognized as the most promising approach [5].

However, hardware-software (HW/SW) co-design shows several challenges for the designer. Not only the application has to be partitioned into software and hardware tasks, but also the communication between them has to be efficiently managed.

In this paper, we evaluate two different design strategies for the design and the implementation of real-time embedded systems for image and video processing based on a FPGA SoC. Each of the two designs presented here has its own strengths and weaknesses. The former shows an extreme flexibility and a moderate performance, whereas the latter requires more design efforts but allows much higher speed performance to be reached. As a case study, a complete image processing architecture, which includes image capturing from a webcam, Sobel filtering and output visualization on a monitor, has been implemented. All experiments have been performed by using a Zedboard

 ISBN: 978-1-61208-585-2

equipped with the Xilinx Zynq XC7Z020-CLG484 SoC. Performance measurements revealed that the frame rates of such designed embedded systems range between 4 and 52 times the frame rate attainable by a pure software counterpart. The rest of the paper is organized as follows. In Section II, a brief background and the most relevant state of the art related works are reviewed. The first evaluated architecture is introduced in Section III, whereas Section IV describes an example of application to perform fair comparisons. A different design approach is then investigated in Section V. Finally, some conclusions are drawn in Section VI.

II. BACKGROUND AND RELATED WORKS

The essence of embedded systems design is implementing a specific set of functions in order to accomplish constraints on performance, costs, emissions, power consumption, etc [6]. Figure 1 shows the typical architecture of a generic embedded system. In general, one or more programmable processing units (CPUs) are used. Depending on the application domain, the embedded systems can have external memory blocks, communication interfaces and several I/O peripherals. While CPUs are traditionally software programmed, custom application specific circuits accelerate more time consuming processes. The first preliminary design step is the efficient HW/SW partitioning of the target application. It consists of splitting the application into computational tasks to be executed either by software routines or by hardware modules. Depending on this partitioning, speed performances and design complexity can be traded off. The generally used approach is to profile the application by means of specific CAD tools [7] in order to indentify its computational loads. Speed performances can be optimized allocating the processing of the most time-consuming tasks to custom hardware accelerators. The remaining non-critical tasks are executed by software routines running on the host general purpose processor. To avoid communication bottlenecks, in a similar approach, the host processor and the hardware accelerators must exchange and transfer data to each other with high throughput and low latency.

The emerging approach based on heterogeneous programmable SoCs is stimulating many application fields [8]. In this section, a group of significant related works are briefly reviewed.

A prototyping environment for heterogeneous CPU/FPGA systems is described in [9], in which a host machine is coupled to a Xilinx Virtex 6 FPGA through the PCI Express Bus. As shown in [10], the limited bandwidth of the communication bus reduces the achievable performances.

A Cadence virtual platform modeling the Xilinx Zynq-7000 SoC is adopted in [11] to implement an Adaptive Cruise Control Unit. This virtual prototyping environment allows using the SystemC language for the portable implementation of software and hardware modules, thus

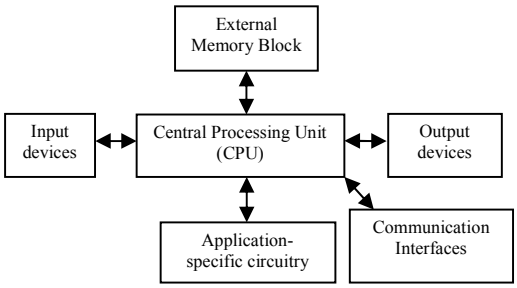

Figure 1. The typical structure of an Embedded System.

avoiding VHDL designs and speeding up the simulation of the overall system. A similar approach is adopted also in [12], where a very high abstraction design approach, based on the use of OpenCV and SystemC, is proposed as an efficient strategy to design embedded systems.

A study about the portability of the OpenCL programming model is, instead, presented in [13]. OpenCL is a framework for targeting heterogeneous platforms based on the C/C++ language.

This approach allows a HW/SW co-design that is independent of the adopted hardware platform to be obtained. Thus, the code can be easily re-targeted to different platforms. In [13], experiments conducted by using the Altera SDK for OpenCL (AOCL), however, demonstrate that the same test code performs differently on different platforms, thus requiring specific optimizations.

In [14], several real-time image processing algorithms are implemented on a Zynq-based hardware platform. This study exploits a task partitioning of the target application based on performances improvements. Linux operating system is hosted on the ARM CPU inside the Zynq to easily manage the video acquisition by software routines. An efficient communication strategy between hardware accelerators and the host CPU is realized through the AMBA Advanced Extensible Interface (AXI).

With the main objective of reducing the time-to-market of the developed system, the approach described in [15] exploits the Xillybus IP core to guarantee a fast communication between hardware and software components of the overall system. To this aim, the communication is managed by software thanks to some useful functions included in the Xillybus library.

In the next section, we evaluate such an approach as a generally valid support to design heterogeneous embedded system architectures for real-time image processing. The Xillinux open source operating system is hosted on the CPU and it manages the communication with the hardware implemented into the FPGA portion of the Zynq chip as a regular peripheral.

III. THE XILLYBUS-BASED PLATFORM

The main target of the platform described below is to furnish an efficient hardware support to develop real-time

 ISBN: 978-1-61208-585-2

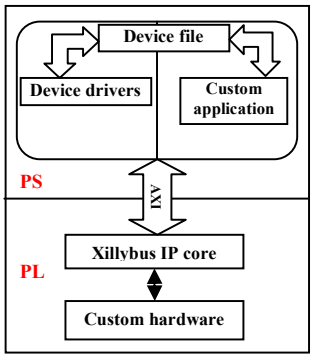

Figure 2. Xillybus-based platform. The PS section hosts the OS components, whereas the PL section implements hardware accelerators and Xillybus IP core.

image and video processing applications in embedded systems, with reduced implementation time.

The platform evaluated in this section is structured as depicted in Figure 2. The Zynq processing system (PS) consists of a dual core ARM Cortex – A9 processor, while the programmable logic (PL) is based on the Artix-7 FPGA fabric for minimizing power consumption.

The ARM processor is able to host OSs such as Linux, Real Time Operating System (RTOS), Windows, etc. The Zedboard is also equipped with 512 MB DDR3 memory and a 256Mb 4-bit SPI Serial NOR Flash memory. The latter supports speeds up to 400Mbps and hence it is suitable for storing boot loaders and kernel of one of the above OSs.

The system detailed below exploits a set of precompiled sub-systems, namely the Xillybus package, able to facilitate the communication tasks between the PS, the external peripherals and the accelerators. The Xillybus package makes also available the Xillinux open source OS that is a complete graphical Ubuntu 12.04 LTS-based Linux distribution, well suitable for rapid development of mixed software/logic designs [16]. It is a collection of software tools that supports roughly the same capabilities of a personal desktop computer running Linux. Xillybus distribution comes with two different synthesizable cores: the XillyLite IP core that allows a simple direct address/data transfer; and the Xillybus IP core that allows data streams to be transferred to/from the custom hardware accelerator [17].

The designed architecture is illustrated in Figure 3. It can be observed that a XillyLite core is used to access a block RAM, whereas the Xillybus core is adopted to transfer data from the PS to/from the custom hardware accelerator [18]. All IP cores are also connected to the PS by an AXI-Lite interface.

When connected to the Xillybus IP core, the hardware accelerators can be accessed by the PS like a common peripheral, which communicates with the OS through specific device drivers.

The interface between the software drivers and the software application is represented by the device files provided by Xillybus. These files can be opened, read and written like any files inside the user space application, so it is possible to implement a high level abstraction for PS-PL communication.

The PS manages the data transfer to capture the frames from a webcam through the USB port and to store them into the DDR3 external memory. Other memory accesses, related to the data transfers to/from the custom hardware accelerator, are governed by the Xillybus IP core through the high performances ports in the PS section. The acquisition operation is easily implemented in software by using video libraries and camera drivers. Whereas, the PL was used to hardware implement the following components:

- The Xillybus IP core that communicates with the PS through the AXI full and AXI Lite interfaces;
- The XillyLite IP core that communicates with the PS through the AXI Lite interface;
- The custom hardware accelerator;
- Two FIFOs, used as input and output interfaces between the Xillybus core and the custom hardware accelerator;
- A VGA controller connected to an external monitor that displays the output of the custom hardware accelerator stored in the DDR3 external memory.

In the designed architecture, the processor works as master during the configuration of the VGA controller, the XillyLite IP core and the Xillybus IP core. This configuration corresponds to a control signals transfer, needed to inform the hardware IP cores about the image resolution, the DDR memory addresses etc.

Each data transfer through an AXI interface occurs as summarized in the following:

- In a read process, the slave device address is sent by the master interface over the read address channel. Then, the addressed slave interface sends the corresponding data over the read data channel to the master.
- In a write process, the master interface sends the slave device address to which the data is to be written and corresponding data. On successful write at the slave interface, the slave sends a response over the write response channel to flag the transfer completion.

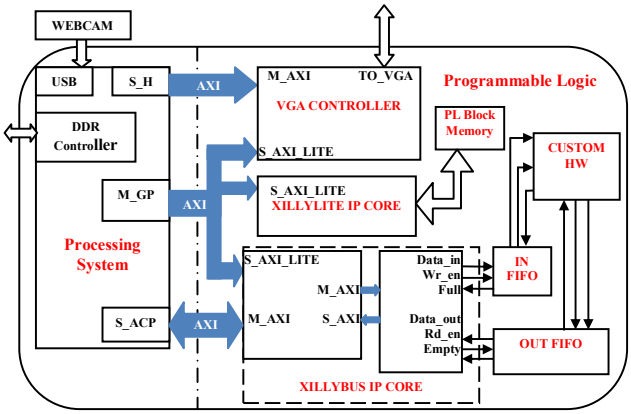

Figure 3. Architecture used within the ZedBoard to evaluate the Xillybus-based platform.

An AXI Interconnect IP core multiplexes the access by the master to the three slaves and the Xillybus IP core acts as an interface between the PS and the custom hardware accelerator. This architecture can be easily customized to perform virtually any image and video processing algorithm without re-design either the top-level architecture or the interface modules required to acquire input images/videos and to display/store the resulting frames.

Data transferred to/from the PS from/to the hardware accelerator flows through the input and output FIFOs, as shown in Figure 3. FIFOs can be configured according to the applications requirements, but they must comply with the constraints of the device drivers provided by Xillybus. As an example, the Xillybus drivers can be configured to transfer 8, 16 or 32-bit data words so the data width of the FIFOs must be set accordingly. The *wr_en*, *rd_en*, *full* and *empty* control signals manage the synchronization between the Xillybus IP core and the hardware accelerator.

The complete data flow implemented within an embedded system designed by using our platform is described in the following:

- The PS converts the RGB frame, captured by the webcam, into the 8-bit grayscale format and, subsequently, it transfers the pixels to the external DDR3 block memory. The frame is transferred from the DDR3 memory to the PL through the Xillybus interface. This operation is performed by the software routine running on the PS, which communicates with the Xillybus drivers through the available “open” and “write” functions applied on the specific device file, as shown in Figure 4. Then, the pixel transfer from PS to the Xillybus IP core occurs through the high performance S_AXI_ACP port. As a response, the Xillybus IP core activates the write enable (*wr_en*) signal of the input FIFO.
- If the input FIFO is not empty and the hardware accelerator is ready to receive the input pixels, the read enable (*rd_en*) signal is asserted and a stream of pixels is sent to the hardware accelerator.
- When valid data is available on the output port of the hardware accelerator, the latter asserts the write enable (*wr_en*) signal of the output FIFO, which receives a stream of data produced by the user-defined computational logic. The output data stream continues until the output FIFO becomes full. If this condition occurs, the output FIFO asserts its *full* signal and the hardware accelerator temporarily stalls the transfer.

```
int fdw;
unsigned char *buffer;
//Open Xillybus interface to transfer data from PS to PL
fdw=open(“/dev/Xillybus_write_device”, O_WRONLY);
write(fdw, buffer, sizeof(buffer));
//Close Xillybus PS-to-PL interface
close(fdw);
```

Figure 4. Use of "open" and "write" functions in the software routine.

```
int fdr;
unsigned char *buffer;
//Open Xillybus interface to transfer data from PL to PS
fdr=open(“/dev/Xillybus_read_device”, O_RDONLY);
read(fdr, buffer, sizeof(buffer));
//Close Xillybus PL-to-PS interface
close(fdr);
```

Figure 5. Use of "open" and "read" functions in the software routine.

- The software running on the PS invokes the “open” and “read” functions of the Xillybus driver, as described in Figure 5, so data stored in the output FIFO is transferred to the DDR3 through the S_AXI_ACP/Xillybus connection. In hardware, this operation corresponds to assert the *rd_en* signal of the output FIFO.
- The output image is finally transferred from the DDR3 to the VGA controller that is connected to an external monitor. The PS is involved in this operation only to send the control signals to the VGA controller through its M_AXI GP port. Pixels to display are transferred from the DDR3 to the VGA controller through the high performance S_AXI_HP port of the PS. The latter is not involved during the data transfer so it can run the next software routine.

In the proposed design support platform, the Xillybus IP core and the custom hardware accelerator work with the same clock, so the write/read operations to/from the input and output FIFOs occur at the same rate. The clock is produced by the PS with a frequency of 100MHz, which is the highest frequency supported by the Xillybus IP core [19]. The usage of synchronous streams is the preferred choice when tight synchronization is needed between the software running on the PS and the hardware implemented in the PL. However, in order to increase performances, multiple clock domains can be adopted if the hardware accelerator can run at clock frequencies higher than 100MHz. In such a case, asynchronous FIFOs with different input and output clock frequencies have to be employed. In particular, the write (read) operation into (from) the input (output) FIFO is performed at the Xillybus clock rate, whereas the write (read) operation into (from) the output (input) FIFO is performed at the clock rate of the hardware accelerator.

IV. THE CASE STUDY: A SOBEL FILTER IMPLEMENTATION

As an example of application, the above described design platform has been used to implement an embedded system which filters digital images. The 3×3 Sobel filter [20] is hardware implemented and applied to 320×240 pixels frames captured by the external camera. Image filtering has been chosen as the case study since it has a computational complexity sufficiently high to highlight the advantages offered by the HW/SW co-design over the all-software counterpart.

The hardware accelerator has been developed with the Vivado High Level Synthesis (HLS) tool that allows describing the hardware circuit in a high level programming

 ISBN: 978-1-61208-585-2

TABLE I. RESOURCE UTILIZATION.

Sobel accelerator			Custom FIFOs			Xillybus IP core			VGA Controller IP core			XillyLite IP core		
LUTs	FFs	32K BRAMs	LUTs	FFs	32K BRAMs	LUTs	FFs	32K BRAMs	LUTs	FFs	32K BRAMs	LUTs	FFs	32K BRAMs
212 0.4%	172 0.1%	1 0.7%	90 0.1%	96 0.1%	1 0.7%	3011 5.6%	2690 2.5%	1 0.7%	499 0.9%	779 0.7%	1 0.7%	67 0.12%	94 0.1%	0 (0%)

language (C++) and converting the code into a synthesizable RTL description. Input and output interfaces of the custom hardware accelerator have been configured as FIFOs, in order to guarantee the compatibility with the two FIFOs connected to the Xillybus IP core. The FIFOs have a data width and a depth of 8 bit and 2048 words, respectively.

Table I summarizes the overall FPGA resources utilization of the implemented architecture. The number of the total used look-up tables (LUTs) is very limited, about 7% of the LUTs available in the XC7Z020-CLG484 chip; the number of required flip-slops (FFs) and 32Kbyte block RAMs (32K BRAMs) is even lower (3.5% and 2.8%, respectively).

The software application, running on the PS, exploits OpenCV library functions [21] to manage the input frames captured by the USB camera. The input pixels are converted from the RGB to the 8-bit grayscale format and transferred to the DDR3 memory. The software application is responsible for transferring the pixels to the Xillybus IP core, retrieving the output pixels from the hardware accelerator and storing them to the DDR3 memory. Finally, the software application starts the data transfer from the DDR3 to the VGA controller. Figure 6 shows two output video frames obtained by the implemented architecture.

To measure execution time of each task, the appropriate software timing library has been used. Since the data transfer through the Xillybus can be performed by varying the number of bytes transferred at each write/read function call, we evaluated the execution time as a function of the bytes packet size. As depicted in Figure 7, the total execution time drastically decreases when the packet size increases. But, the minimum execution time of about 118.3 ms is reached for a packet size of ≈5000 bytes and it is maintained until 9600 bytes.

Figure 6. Some input and output video frames.

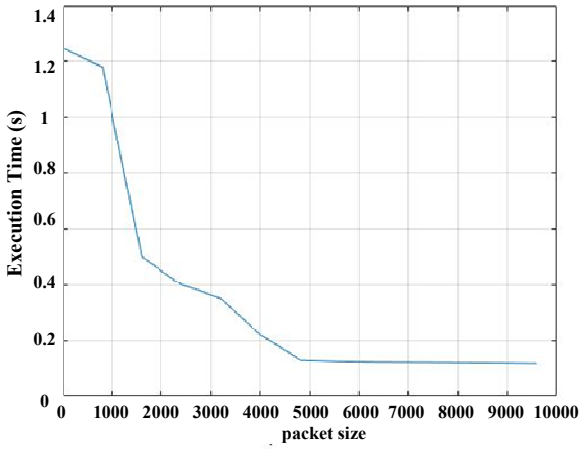

Figure 7. Execution time vs. the packet dimension.

TABLE II. EXECUTION TIMES.

Hardware Sobel filter	Software application	Communication	All-software Sobel filter
2.3 ms	100 ms	16 ms	490 ms

This result suggests to adopt transfer of ≈5000 bytes each, because this also limits the depth of the input and output FIFOs.

Table 2 shows the timing breakdown that is split into three main contributions: the hardware processing time (Sobel filtering), the PL-PS communication time ("write, "read" and "open" of the Xillybus driver) and the remaining software execution time (RGB to grayscale conversion and data streaming from/to the DDR3 managed by the PS) per frame.

As expected, the software execution time represents the highest contribution, mainly due to the OpenCV functions for the management of input frame, the format conversion and the output frame visualization. The hardware processing and the communication between PS and Xillybus IP core account only for the 15.5% of the overall execution time.

In order to estimate the speed-up obtained by the custom hardware accelerator, a pure software routine performing the same Sobel filtering has been characterized. The latter has been executed by the ARM processor hosted in the PS, which operates at a 666.66 MHz running frequency. Measurements reported in Table 2 show the benefits obtained by the heterogeneous design approach. A gain of about 4x has been achieved.

Even though a direct comparison between our results and those reported in [15] cannot be performed, due to the different user application, a brief discussion is appropriate. In [15], a Xillybus-based platform performing the Harris

 ISBN: 978-1-61208-585-2

Corner Detection function on 512x384 images has been implemented and evaluated. When the PL is clocked at 100 MHz and the 32-bit Xillybus software interface is adopted, the total communication and hardware processing time is about 15 ms. The latter is approximately 3.3ms lower than result reported in Table 2, which instead has been obtained adopting the 8-bit Xillybus software interface. If the 32-bit Xillybus software interface is used in the architecture of Figure,3, the communication time is reduced correspondingly.

V. THE VDMA-BASED ALTERNATIVE

The high level design approach used above employs a ready-to-use communication solution between hardware and software components. Due to this, it significantly reduces design efforts, the development time and the hardware design expertise required for realizing a complete embedded system for video processing applications. Of course, such a design strategy negatively impacts the overall speed performances. In particular, some considerations can be done in reference to the Xillybus bandwidth.

The FIFO configuration provided by Xillybus has a maximum bandwidth of about 200 MB/s for each transfer direction [17]. On the contrary, the Zynq PS high performance ports are able to access the DDR3 memory achieving a bandwidth of 1600 MB/s for a 64-bit transfer at 100 MHz clock rate [22].

In this section, we examine an alternative design, based on the direct use of Video Direct Memory Access (VDMA) IP cores [23]. Using this approach, much more architectures design knowledge and digital system debugging practice are required. The VDMA is a soft core, which provides high bandwidth access to external memory and video processing IP cores with AXI-Stream interface. The architecture designed in accordance with this strategy is illustrated in Figure 8. In this case the PS does not support an OS, thus the system is oriented to bare-metal application architectures. An OmniVision OV7670 CMOS Camera has been connected through an I2C interface and an appropriate frame capture control sub-system is required. The VDMA0 transfers captured frames to the DDR3 and, then, after the elaboration, from the DDR3 to the VGA display port. The VDMA1 transfers the video stream to the custom hardware accelerator that performs the specific video algorithm. After that, the VDMA1 writes back the filtered results into the DDR3 memory.

Using High Performance AXI ports to access the external memory allows the computational load of the processor to be significantly reduced. Furthermore, by using two different High Performance ports, parallel operations to/from the DDR3 can be performed, thus obtaining a further considerable performance improvement. In fact, a new captured frame can be stored, or a result frame can be displayed, while the VDMA1 transfers the pixel stream to/from the hardware accelerator.

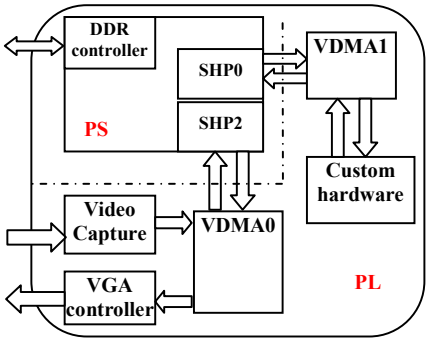

Figure 8. Architecture used within the ZedBoard to evaluate the VDMA-based platform.

Due to the overlap between the two phases above mentioned, this architecture reaches a processing rate much higher than the structure exploiting the Xillybus IP Cores. When the same Sobel filter accelerator is implemented within this structure, a total execution time of only ≈9.2 ms is achieved, thus leading to an overall performance ~13 times higher. This result has been obtained with a clock rate of 100MHz for the PL section, while the Video Capture IP core operates at 30 frames per second in VGA resolution.

VI. CONCLUSIONS

In the development of an embedded system based on heterogeneous SoCs, of course, the first important design step is the efficient HW/SW partitioning of the target application. After that, on the basis of the design environment, several other significant choices have to be done. When only high level description is desired, several precompiled supports could be of great help. As shown in this paper, an almost complete solution is offered by the Xillybus package that contains appropriate communication sub-modules and a light OS.

We designed a test architecture to evaluate the speed improvement attainable with such supports and measured a speed up of ≈4 times with respect to a pure software typical image processing elaboration. Such an approach allows very easy interface between the designed architecture and peripherals.

On the contrary, when the speed performance is the main concern, a direct and on-purpose design of the entire architecture is preferable. In such a case, a further x13 speed up has been observed, but at the expense of much more design effort and verification time.

ACKNOWLEDGMENT

Authors wish to thank Dr. Fabio Frustaci and Dr. Giovanni Staino for the helpful discussions during the present work.

REFERENCES

[1] W. Wolf, “The Future of Multiprocessor Systems-on-Chips” Proceedings of the 41st annual Design Automation Conference (DAC ’04), pp. 681-685 , June 7-11 2004.

[2] B. Roux, M. Gautier, O. Sentieys, and S. Derrien, "Communication-Based Power Modelling for Heterogeneous Multiprocessor Architecture", IEEE 10th International Symposium on Embedded Multicore /Many-core Systems-on-Chip (MCSoC 2016), pp. 209-216, Sep 2016.

[3] J.A. Kalomiros and J. Lygouras, "Design and evaluation of a hardware/software FPGA-based system for fast image processing", Microprocessors and Microsystems, vol. 32, issue 2, March 2008.

[4] R. J. Petersen and B. L. Hutchings, "An assessment of the suitability of FPGA-based systems for use in digital signal processing". In: Moore W., Luk W. (eds) Field-Programmable Logic and Applications. FPL 1995. Lecture Notes in Computer Science, vol. 975, pp. 293-302. Springer, Berlin, Heidelberg

[5] J. Teich, "Hardware/Software Codesign: The Past, the Present, and Predicting the Future", Proceedings of the IEEE, vol. 100, Issue: Special Centennial Issue, pp. 1411-1430, May 13 2012.

[6] A. Sangiovanni-Vicentelli and G. Martin, "Platform –based Design and Software Design Methodology for Embedded Systems", IEEE Design & Test of Computers, IEEE press, vol. 18, issue 6, pp. 23-33, Nov/Dec 2001, doi: 10.1109/54.970421.

[7] R. Ernst, "Codesign of Embedded Systems: status and trends", IEEE Design & Test of Computers, vol. 15, issue 2, pp. 45-54, Apr-Jun 1998, doi: 10.1109/54.679207.

[8] D. Andrews, D. Niehaus, and P. Ashenden, "Programming models for hybrid CPU/FPGA chips", Computer, IEEE press, vol. 37, issue 1, pp. 118-120, Jan. 2004, doi: 10.1109/MC.2004.1260732.

[9] G. Afonso, R.B. Atitallah, A. Loyer, J. Dekeyser, N. Belanger, and M. Rubio, "A prototyping environment for high performance reconfigurable computing" 6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), IEEE press, pp. 1-8, August 2011, doi: 10.1109/ReCoSoC.2011.5981497.

[10] Y. Li, X. Zhao, and T. Cheng, "Heterogeneous Computing Platofrom Based On CPU+FPGA and Working Modes", 12th International Conference on Computational Intelligence and Security (CIS), IEEE press pp. 670-672, December 2016, doi: 10.1109/CIS.2016.0161.

[11] P. Wehner, M. Ferger, D. Göhringer , and M. Hübner, "Rapid Prototyping of a Portable HW/SW Co-Design on he Virtual Zynq Platform using SystemC", IEEE 26th International SOC Conference (SOCC), IEEE press pp. 296-300, September 2013, doi: 10.1109/SOCC.2013.6749704.

[12] J. Anders, M. Mefenza, C. Bobda, F. Yonga, Z. Aklah, and K. Gunn, "A hardware/software prototyping system for driving assistance investigations" in Journal of Real-Time Image Processing , vol. 11, issue 3, pp. 559-569, March 2016.

[13] S.O. Ayat, M. Khalil-Hani, and R. Bakhteri, "OpenCL-based Hardware-Software Co-design Methodology for Image Processing Implementation on Heterogeneous FPGA Platform", 2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE), IEEE press pp. 36-41, November 2015, doi: 10.1109/ICCSCE.2015.7482154.

[14] M.A. Altuncu, T. Guven, Y. Becerikli, and S. Sahin, "Real-Time System Implementation for Image Processing with Hardware/Software Co-design on the Xilinx Zynq Platform", International Journal of Information and Electronics Engineering, vol. 5, no. 6, pp. 473-477, November 2015, doi: 10.7763/IJIEE.2015.V5.582.

[15] I. Stratakos, D. Reisis, G. Lentaris, K. Maragos, and D. Soudris, "A Co-Design Approach For Rapid Prototyping Of Image Processing on SoC FPGAs"Proceedings of the 20th Pan-Hellenic Conference on Informatics (PCI '16), November 2016, ISBN: 978-1-4503-4789-1, doi: 10.1145/3003733.3003797.

[16] Getting started with Xillinux for Zynq-7000 EPP v. 1.3 [Online]. Available from: http://xillybus.com/downloads/doc/xillybus_getting_started_zynq.pdf

[17] B.M. Kambalur, K. Kumar, and K.S. Athrey, "A Study of Implementing Custom Application on Zynq AP SoC using Xillybus IP Core", ITSI Transactions on Electrical and Electronics Engineering (ITSI-TEEE), vol.. 2 pp. 35-37 , issue 5-6, 2014.

[18] Getting started with the FPGA Demo Bundle for Xilinx v. 2.6 [Online]. Available from: http://xillybus.com/downloads/doc/xillybus_getting_started_xilinx.pdf

[19] Xillybus FPGA Designer's Guide v. 2.0 [Online]. Available from: http://xillybus.com/downloads/doc/xillybus_fpga_api.pdf

[20] N. Kanopoulos, N. Vasanthavada, R., and L. Baker, "Design of an Image Edge Detection Filter Using the Sobel Operator", IEEE Journal of Solid-State Circuits, vol. 23, issue 2, pp. 358-367, Apr 1988, doi: 10.1109/4.996.

[21] About OpenCV [Online]. Available from: http://opencv.org/about.html

[22] Zynq-7000 All Programmable SoC Technical Reference Manual, UG585 (v. 1.11) September 27, 2016 [Online]. Available from: https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf

[23] C. Kohn, "Partial Reconfiguration of a Hardware Accelerator on Zynq-7000 All Programmable SoC Devices" XAPP1159 Xilinx Jan. 21, 2013.