High Level Modeling and Hardware Implementation of Image Processing Algorithms Using XSG

Kamel Messaoudi, Hakim Doghmane, Hocine Bourouba, E. B. Bourennane, and S. Toumi

Abstract Design of Systems-on-Chip has become very common especially with the remarkable advances in the field of high-level system modeling. In recent years, Matlab also offers a Simulink interface for the design of hardware systems. From a high-level specification, Matlab provides self-generation of HDL codes and/or FPGA configuration codes while providing other benefits of easy simulation. In addition, a large part of the Systems-on-Chip use at least one image processing algorithm and at the same time border detection is one of the most used algorithms. This paper presents a study and a hardware implementation of various algorithms of borders detection realized under Xilinx System-Generator. The various algorithms are implemented using Xilinx FPGA device, the simulation and synthesis results are also compared. We use the Xilinx Zed-Board for physical implementations in-the-loop.

Keywords Edge detection · Hardware implementation · Image processing · VHDL · Xilinx system generator · Xilinx Z-Board

1 Introduction

In recent years and with the remarkable advances of powerful and low-cost processing chips, applications of signal processing (and in particular image processing) have grown dramatically in importance [1]. In-fact, media information such as audio, images, and video have come to be necessary for various computer applications [2] such as medical, video surveillance, target recognition, robotics application, etc…

K. Messaoudi (✉) · E. B. Bourennane · S. Toumi
Electrical Engineering Department, Mohemed Cherif Mssaadia University, Souk-Ahras, Algeria
e-mail: k.messaoudi@univ-soukahras.dz

H. Doghmane · H. Bourouba
Laboratory of Inverse Problems, Modeling, Information and Systems (PIMIS), 8 Mai 1945 University, Guelma, Algeria
e-mail: doghmane_hakimaz@yahoo.fr

H. Bourouba
e-mail: bourouba2004@yahoo.fr

S. Bououden et al. (eds.), *Proceedings of the 4th International Conference on Electrical Engineering and Control Applications*, Lecture Notes in Electrical Engineering 682,
https://doi.org/10.1007/978-981-15-6403-1_71

Image processing has a strong mathematical basis where the quality is enhanced by using various processing techniques. The main objective of image processing is to improve the quality of the images for human interpretation and analysis [3].

Several basic processes are applied to a digital image defined in a digital computer as being a matrix of pixels. Acquisition, Enhancement, Restoration, Segmentation and Analysis are the steps needed by just about every application which involves image processing [4]. Edges are one of the most significant features for image processing and many computer vision applications. In various image processing algorithms, the filtering operation (and in particular the edge detection) is generally the first step used in order to eliminate noises and to reduce the amount of data. In digital image, the edge defines the object boundaries within the image and occurs when discontinuities present in the pixel values. Edge detection is a preprocessing step for many image processing algorithms such as image enhancement, image segmentation, object tracking, object recognition and image/video coding [3].

Currently, Matlab-Simulink is enriched with features and templates that simultaneously enable data processing (using fixed-point precision) and ensure self-generation of synthesizable VHDL codes for FPGA targets. In addition to the easy simulation provided by Matlab-Simulink and using the new Xilinx System-Generator (XSG tool), we can directly implement the proposed Simulink models on FPGA through the various simulation and co-simulation steps, the Input/Output (I/O) assignment, the temporary and spatial constraints and at the synthesis and implementation.

This paper presents a study with hardware implementations of edge detection algorithms using the XSG tool. The idea is to compare Matlab-Simulink image processing results (using floating point precision) with XSG image processing results (using fixed-point precision). The idea is also to implement advanced image processing algorithms and in particular edge detection methods. Using the new "in the loop" technique, a first part of the proposed model is simulated under Simulink using the main processor and a second part is implemented in real time on FPGA device. Matlab-Simulink will provide real-time communication and links between both parts.

The rest of this paper is organized as follows: we first define image processing and edge detection in Sect. 2. The XSG design flow is presented and discussed in Sect. 3. In Sect. 4, we present and detailed the proposed hardware implementation of edge detection algorithms. Simulation and synthesizes results are discussed in Sect. 5. Finally, we present the conclusions and outline future work in Sect. 6.

2 Image Processing and Edge Detection

Image processing is a very vast field that has known, and still knows, an important development for a few decades. By digital image processing is meant all the techniques enabling a digital image to be modified in order to improve the visual quality or to extract the relevant information from it.

Image processing is a discipline widely used in several fields namely: medicine, military field, target detection, robotics, etc. This discipline is highly sought also in the field of automation and robotics where important decisions are often taken from the images processed by various algorithms.

There are number of well defined processes which go to make up a typical image application. Acquisition, Enhancement, Restoration, Segmentation and Analysis are the steps needed by just about every application which involves image processing [2]. Edges are one of the most significant features for image processing and many computer vision applications. Edge detection is one of the classical studies since it is perquisite thing prior to perform any image processing operation. It has provoked interest among research fraternity since several years [3].

A. *Edge detection*

Edge detection is used as a pre-processing step for many image processing algorithms such as image enhancement, image segmentation, tracking and image/video coding. This is a basic operation in image processing, and refers to the process identifying and locating sharp discontinuities in an image; the discontinuities are abrupt changes in pixel intensity which characterize boundaries of objects in a scene [5].

In the literature, various gradient based edge detection algorithms are proposed such as Robert, Prewitt, Sobel and Canny (as example). These operators are firstly linear and secondly use simple convolution masks. The elements of these masks are often in the set $\{-2, -1, 0, 1, 2\}$ which makes easier the convolution operations in the mathematical expressions of the filters. Indeed, in a mathematical operation the multiplication by $(-1, 0$ and $1)$ can be suppressed and the multiplications by 2 and by -2 can be realized using additions or more simply by left offsets. This is an advantage in a software and/or hardware implementation generally to avoid the expensive multiplication operations (hardware resources and computing time). The ultimate goal is to speed up calculations of mathematical expressions in filters especially with larger amounts of data in the images.

B. *Hardware implementation of edge detection algorithms*

Most implementations of image processing algorithms are software and run on personnel computer and generally do not respect real-time. The purpose of these software implementations is to verify the effectiveness of the proposed algorithms by simulations on synthetic images or real images. These images are extracted from known databases to compare the simulation results.

In the literature, various hardware implementations are proposed for edge detection algorithms in order to ensure real time processing of different tools and applications. In [1, 3, 6], various hardware implementations of gradient based edge detection algorithms (such as Robert, Prewitt, Sobel and Canny) using Xilinx System Generator tool have been proposed. These edge detection algorithms are firstly designed and simulated in Matlab-Simulink and then reproduced using only Xilinx modules (these modules are recently added in Matlab-Simulink). Traditional simulation modules are used to read and display images in Matlab-Simulink (co-simulation).

Designed algorithms are efficiently compiled, synthesized and implemented using FPGA devices.

In this paper, we propose a real-time hardware implementation of some image processing algorithms. In fact, the basic idea of the proposed implementations consists in the realization of a platform of acquisition and display of images while adding software and/or hardware modules. This platform can be used in multiple and varied applications of automatic and industrial computing.

3 XSG Design Flot

Currently, FPGA devices are widely used either to realize prototypes or hardware and real-time implementations of processing algorithms. In the industrial and especially in the real-time control of systems field FPGA devices are also used [7]. Moreover, a new tool called Xilinx System Generator (XSG) is added recently under Matlab-Simulink. XSG is a new design tool that enables the model-based Simulink design environment for hardware implementation (VHDL and RTL levels) and FPGA design. Using the XSG tool, previous experience on hardware implementations and RTL design methodologies are not required. XSG maintains a very harmonious level with Simulink Blockset, but at the same time allows automatic translation of models to synthesizable and efficient hardware implementations using a Xilinx specific Blockset.

The use of Matlab-Simulink allows the easy creation of testbench for simulation and co-simulation of Simulink models and Xilinx models at the same time. In a hardware implementation under Simulink and System Generator various steps are necessary from Description of the algorithm in mathematical terms to the auto generation of HDL code and the synthesis of results.

Figure 1 shows the System Generator design flow. System Generator works within the Simulink model-based design methodology [7]. Firstly, the algorithm of the application can be developed and implemented using the standard Simulink Blockset. Matlab-Simulink uses floating-point numerical precision and without hardware detail. This software implementation can be verified using Simulink simulation results. System Generator can be used to specify the hardware implementation details for the FPGA devices using Xilinx DSP Blockset. The hardware implementation can be verified both using Simulink simulation and System Generator simulation. Simulation results can be compared to propose necessary modifications. System Generator invokes Xilinx Core Generator to generate VHDL code, the NGC file or the Bitstream for the specified FPGA device. Several steps are required before and after the generation of synthesizable codes.

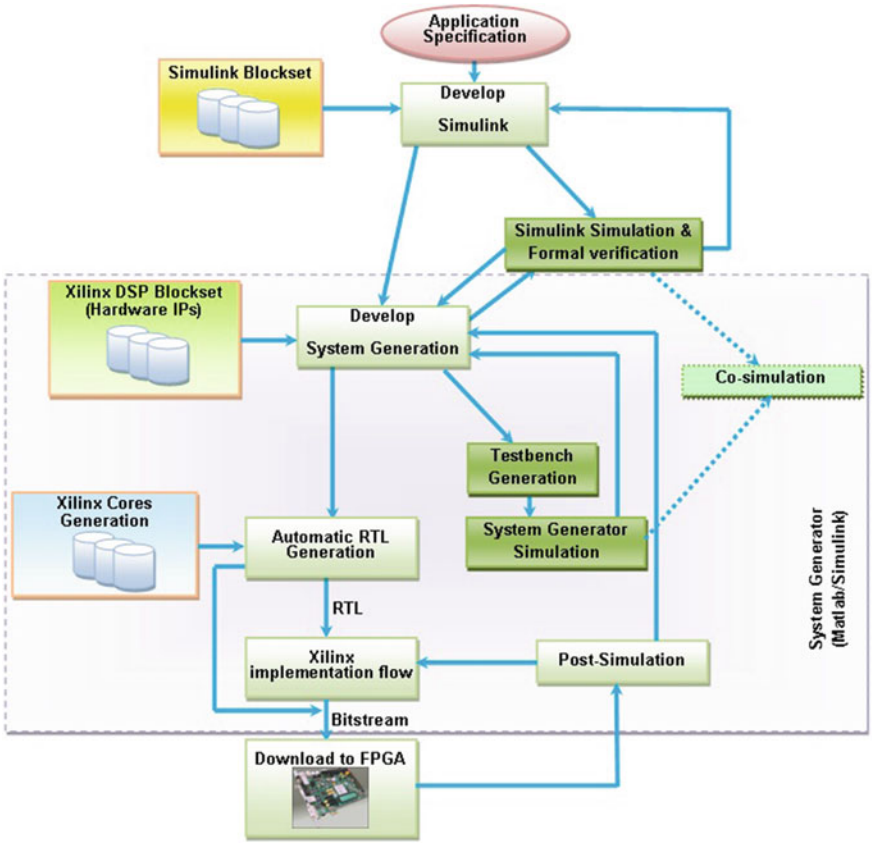


Fig. 1 System-Generator design flow [7]

4 Hardware Implementation of Edge Detection Algorithms

In this section, we will present some hardware implementations of some image processing algorithms using the XSG tool. First, we change the image format (bi-dimensional matrix of pixels) into a one-dimensional signal manipulated by the processing modules provided by Xilinx. Then, we implement the image processing algorithms using the Matlab-Simulink modules. After that, we cross-refer to the implementation of the same algorithms using Xilinx processing modules.

In the second implementation, we copied the same Simulink model of the image processing algorithm using only the Xilinx Blockset. Several tests are carried out for: (i) the good resizing of various variables, (ii) the synchronization of the operating frequencies of the Simulink blocks and (iii) the Xilinx blocks. The purpose of this part is to ensure the transition from floating-point precision (used by Simulink blocks) to fixed-point precision (used by Xilinx blocks) to make the results comparable. Figure 2 shows the flowchart of the proposed main idea.

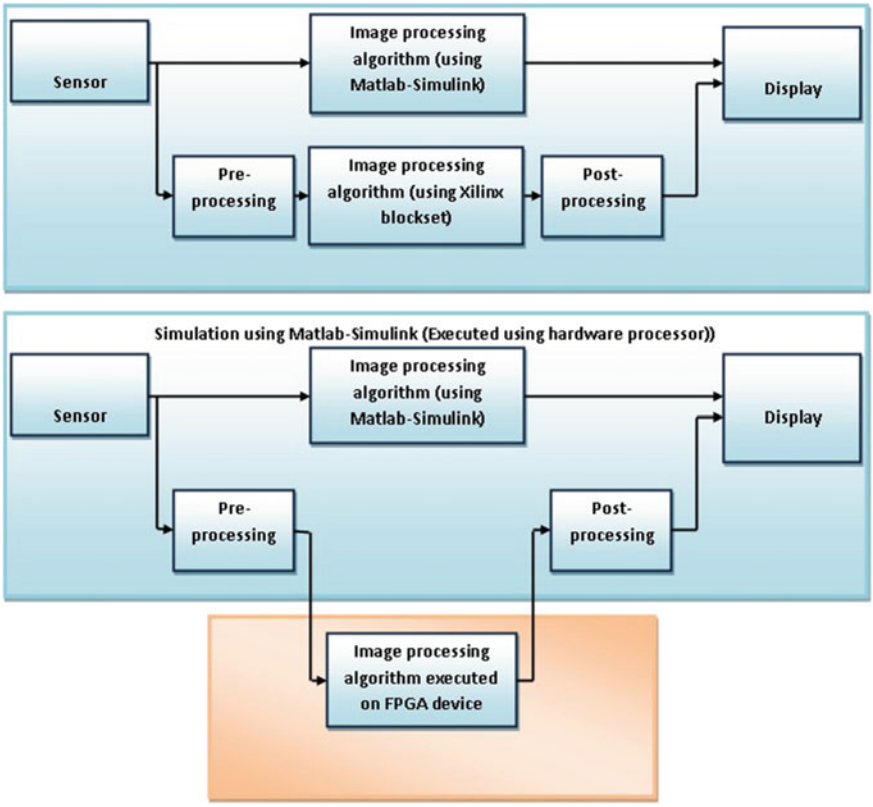


Fig. 2 In-the-loop implementation of the proposed idea

A co-simulation is necessary in order to compare simulation results and to correct errors in the proposed models. Therefore, several adaptations and configurations are necessary for each Xilinx processing module. The advantage of these Xilinx-based models is the automatic transition to synthesizable HDL codes, which are used. We use these HDL codes for FPGA configuration. In addition, for rapid implementation and physical verification [7], these HDL codes are combined with the "in-the-loop" technique.

A. *The processing platform*

Matlab® is software mainly adapted to matrix calculation. A simple scalar number is considered under Matlab as a matrix of size 1×1. An image in Matlab is saved as an $N \times M$ matrix ($N \times M$ pixels, where N and M represent the number of rows and columns, respectively). All operations used in Matlab are applicable on the image matrix. In hardware implementations using an HDL, the details of the operations applied to the simple data are necessary for optimizing the implementations. In order, to manipulate an image, it is necessary to convert the matrix into a vector of pixels. In

this section, we will take advantage of the reading and display tools available in the Matlab-Simulink libraries while taking advantage of the Xilinx System Generator tool for the realization of hardware accelerators in HDL.

The proposed models combined with the image reading and display modules are used for checking the software implementations of the processing algorithms. Then, we propose models based only on Xilinx modules (the XSG library). For the reading and the display of the images for these last models we use the same Matlab modules and we compare the results.

a. *Pre-processing Matlab model*
An image in Matlab is defined as a two-dimensional (2D) arrangement. To meet the requirement of hardware implementations, the image should be pretreated and given as a one-dimensional (1D) vector. The Matlab-Simulink implementation based on a model used for image preprocessing is shown in Fig. 3. In order, to process a 2D image, it is first necessary to calculate the transposed image and then, it is necessary to convert the 2D matrix into a 1D vector using the "convert 2D to 1D" module. Next, we use the "to frame" and "Unbuffer" modules to read the vector with one pixel every time with a predefined scalar data format and at a higher sampling rate.
b. *Post-processing model*
As shown in Fig. 4, the post-processing module is used to convert the output of the image result to the original format. This module is used before the display of the processed image and to compare simulation results. The element modules are practically the inverse modules of the first chain, we start with the conversion of the floating point data using "Data type conversion" then, the "Buffer" and the "Convert 1-D to 2-D" modules for the conversion of pixels in the form of a matrix. Finally, we calculate the transposed matrix with the display of the image (Fig. 4).

B. *The proposed hardware implementations*

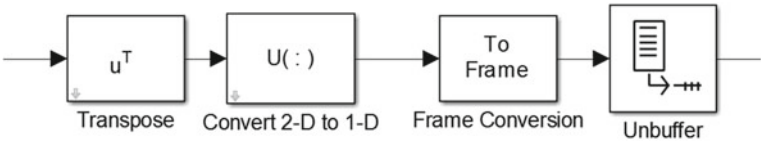


Fig. 3 The used pre-processing model in the proposed implementation

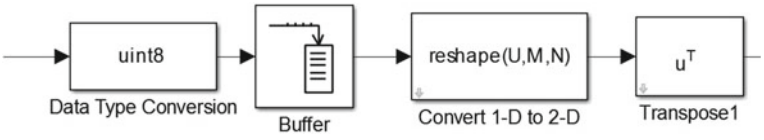


Fig. 4 The used post-processing model in the proposed implementation

In this section, we propose hardware implementations for several filters using Xilinx Blockset. We compare simulation results (co-simulation) and we discuss the synthesis results. Then, we show the "in the loop" execution of the proposed implementations. In order, to optimize these implementations, two large parts are proposed:

- The first part concerning the memory management: this part is utilized for the preparation of pixels concerned by the treatment at each time. In fact, the filtering operation is adapted to a window of size 3 × 3 or 5 × 5 pixels over the entire image matrix.
- The second part is used to define the architecture of the used filter in order to calculate resulting pixel according to the coefficients of the filter window.

a. *Memory management for the used filters*

To ensure the continuity of the pixels subject of processing, we used three (or five) rows of flip-flops according to the number of pixels in the image. If we process an image of size N × M, we use M flip-flops in each line [8]. We have configured these three rows of flip-flops as a FIFO or a shift register.
At the output of this block, we can recover 3 × 3 pixels being processed and each rising edge of the clock a pixel shift is made to recover the 3 × 3 pixels subject of the next processing. By shifting the pixels, we can recover the 3 × 3 pixels processing subjects in the same position, which implies a significant gain in hardware implementations. Indeed, instead of shifting the window of the 3 × 3 pixels in the NxM size image, it is sufficient to shift the pixels of the image and recover the 3 × 3 pixels in the same place (Fig. 5).

b. *The processing module architecture*

This part starts with the point-by-point multiplication of the two windows (the window of 3 × 3 pixels and the window of coefficients). This multiplication is followed by a sum and in some cases by a division. The mathematical formula is given by:

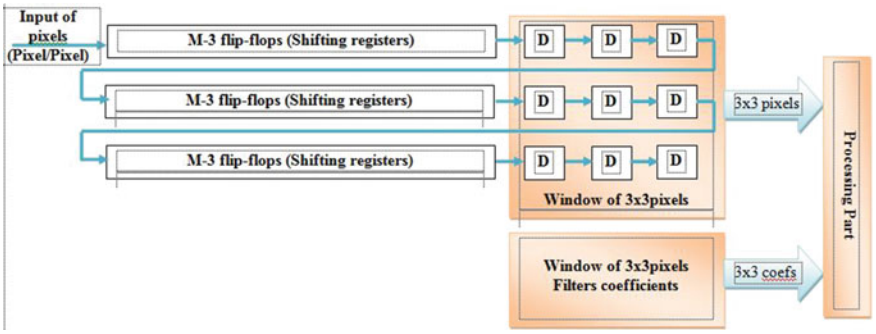


Fig. 5 Block diagram of the memory management strategy used for the implemented filters

$$\text{The pixel result} = \sum_{i=1}^{3} \sum_{j=1}^{3} \text{coefficient}(i, j) * \text{pixel}(i, j) \quad (1)$$

In order to avoid multiplication and division operations (which are costly in terms of FPGA resources consumption) and since most filters have simple coefficients (−1, 0, and 1), this formula is replaced by another formula based on addition, subtraction and shifting operations. At the end of this part, we use a block for the adjustment of data (the value of the pixel to be in the domain 0–255).

C. *The proposed implementation for the 3 × 3 Sobel filter*

Figure 6 shows the proposed hardware implementation under System-Generator for the 3 × 3 Sobel filter. We also notice the presence of the main module (System-Generator) to ensure the automatic conversion of models in HDL level. Several adaptations and data processing's are also necessary to ensure simulation results comparable with the initial filter and to ensure the self-generation of a correct and synthesizable HDL codes.

5 Results and Discussions

A. *Simulation results*

The proposed implementation based on Xilinx Blockset operates using fixed-point precision. The processing results are different from the results of the Matlab filter that operates using floating-point precision. Figures 7 and 8 show the simulation results of the proposed implementations for the Sobel and Prewitt filters applied on two images extracted from Matlab library.

B. Synthesis results

After self-generation operation of the HDL code using System-Generator from a Matlab-Simulink model, a synthesis step is required using one of the Xilinx tools. In this work, we use the Xilinx-ISE tool. This software allows editing, verification, behavioral and functional simulation as well as the synthesis and generation of FPGA configuration files.

In this work, several filters were made under Matlab-Simulink we have subsequently converted these models into VHDL. The synthesis results of these codes are shown in Tables 1. Two options are possible for the synthesis operation: synthesis using RAM blocks or without use of the RAM blocks. These results show the use around 300 slices registers and around 270 slice LUTs as well as the use of 2 RAM block. The number of resources consumed depends on the filter used and the size of the processed image.

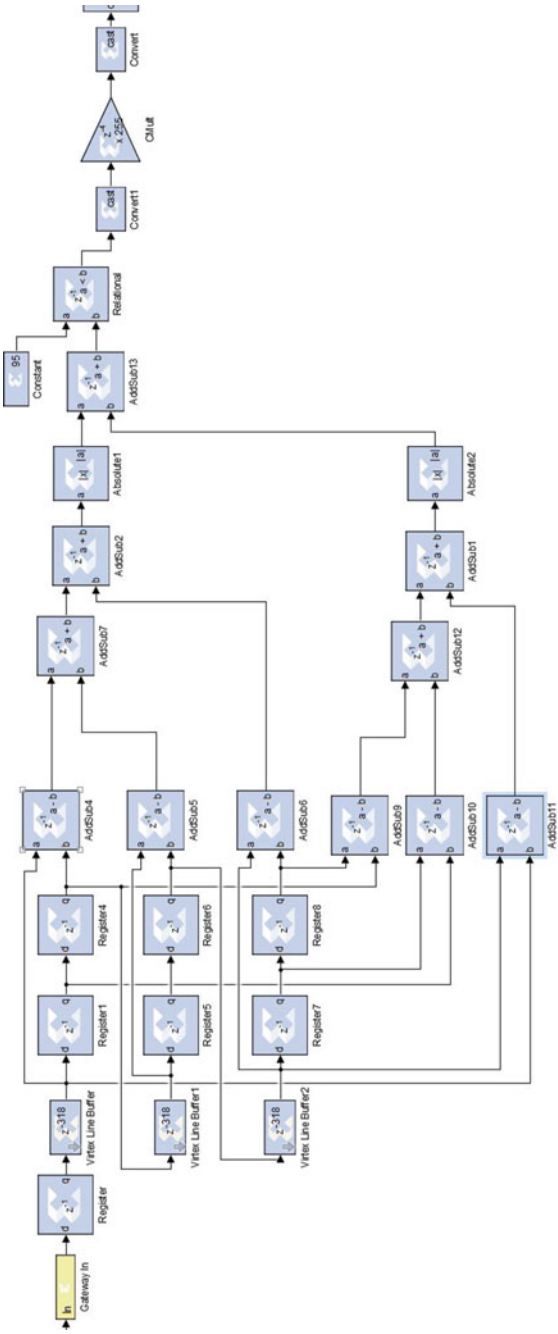
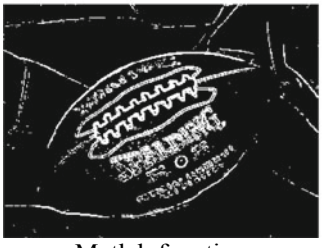


Fig. 6 Hardware implementation of 3×3 Sobel filter using system-generator

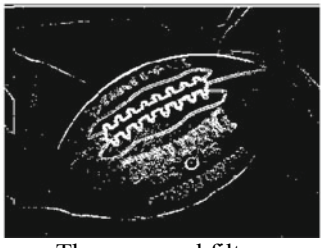
Original image (ball.gif)
320x250pixels

Matlab function
Sobel 3x3

The proposed filter
Sobel 3x3



Matlab function
Prewitt 3x3



The proposed filter
Prewitt 3x3

Fig. 7 Simulation results of filters using Matlab-Simulink (the image mri.tif)

6 Conclusion

In this paper we have proposed hardware implementations of image processing algorithms using Xilinx System Generator. It is in the context of rapid prototyping of real-time implementation of image processing applications using FPGA devices with ever greater integration. As processing algorithms, we have chosen the border detection filters (High-Pass Filters) because of theirs utilizations in several domains and applications. For each filter, we start with a high-level implementation under Matlab-Simulink. Subsequently, we implement the same filter always under Matlab-Simulink but using the Xilinx Blockset. The advantage of this second implementation is that it is automatically convertible into HDL level and synthesizable for an FPGA device.

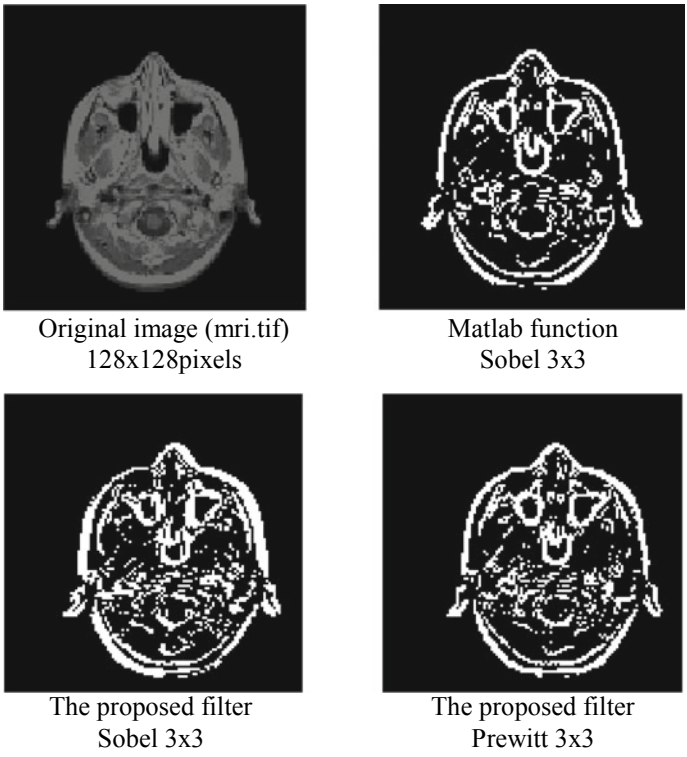


Fig. 8 Simulation results of filters using Matlab-Simulink (the image mri.tif)

Table 1 Synthesis results

Filter & size	Image & size	Slice registers	Slice luts	Block RAM/FIFO
Available on used FPGA device		106400	53200	140
Sobel 3 × 3	football.jpg 320 × 256pixels	343	264	2
Sobel 3 × 3	mri.jpg 128 × 128pixels	355	275	2
Prewitt 3 × 3	football.jpg 320 × 256pixels	249	227	2
Prewitt 3 × 3	Mri.jpg 128 × 128pixels	343	264	2

The use of both implementations allows the co-simulation as well as the comparison of the simulation results. The co-simulation results show the effectiveness of the proposed implementations despite the operation with fixed-point precision. The synthesis results show the consumption of less than 1% of FPGA resources using the Xilinx Zed-Board. As perspectives of this work, we propose to implement other

filters that require more computing time as well as the transition to other types of image processing.

References

1. Elamaran V, Rajkumar G (2012) FPGA implementation of point processes using Xilinx system generator. J Theor Appl Inf Technol 41(2):201–206
2. Alareqi M, Elgouri R, Hlou L (2014) High level FPGA modeling for image processing algorithms using Xilinx system generator. Int J Comput Sci Telecom 5(6), June
3. Avinash GM, Shah AM (2017) FPGA implementation of gradient based edge detection algorithms. Int J Innov Res Comput Commun Eng 5(5), May
4. Bin Othman MF, Abdullah N, Bin Ahmad Rusli NA (2010) An overview of MRI brain classification using FPGA implementation. In IEEE symposium on industrial electronics & Applications (ISIEA), pp 623–628, Oct
5. Rakesh MR (2013) Design and simulation of Matlab/Simulink model for edge detection techniques in image segmentation. Int J Adv Res Electr 2(12):5828–5834
6. Gupta A, Vaishnav H, Garg H (2015) Image processing using Xilinx system generator (XSG) in FPGA 2(6):119–125, Sept
7. Messaoudi K, Yahi A, Messaoudi N, Bourennane EB, Toumi S (2016) Adaptive hardware implementation for the deblocking filter used in H.264/AVC using system generator. In ICESTI'16, International Conference on Embedded Systems in Telecommunications and Instrumentations, Annaba, Algeria, 24–26 Oct 2016
8. Messaoudi K, Bourennane EB, Toumi S, Kerkouche E, Labbani O (2010) Memory requirements and simulation platform for the implementation of the H.264 encoder modules. In IEEE international conference on image processing theory, tools and applications IPTA'10, Paris, pp 133–137, Juillet 2010