Real-Time System Implementation for Image Processing with Hardware/Software Co-design on the Xilinx Zynq Platform

M. Ali Altuncu, Taner Guven, Yasar Becerikli, and Suhap Sahin

***Abstract*—Nowadays, image processing applications are frequently preferred in such fields as industrial automation, security, health, and traffic control in parallel with the developments in technology. The most important criterion for the applications used in these fields is to ensure the system to run at high speed and real time. Thus, FPGAs are commonly used in such applications. In this study, some of the basic real time images processing algorithms are implemented in a FPGA-based development kit. The ZedBoard Znyq-7000 development kit produced by Xilinx Company was used in the study. As a result of grayscale conversion and convolution operations, such applications as edge detection, sharpening and blurring were realized on the images taken by video source. The processed images were viewed by the use of a monitor connected to HDMI output of the development kit. Verilog HDL was used for these operations.**

***Index Terms*—Convolution, FPGA, image processing, real-time systems.**

I. Introduction

Nowadays, the importance of image processing is rapidly increasing in such fields as industrial automation, security, health, and traffic control in parallel with the developments in technology. The most challenging difficulty in applications used in these fields is to make system run real-time [1]. It is not always possible to make the system run real-time by the use of a software used on a general purpose computer since the resources of memory, CPU and peripheral devices in computers are limited. In most image processing applications, dozens of operations are performed on each pixel. That these operations are performed by general purpose processors sequentially leads to negative consequences in terms of both resource consumption and performance [2].

However, FPGAs (Field Programing Gate Arrays) has the capability to operate in a parallel way in terms of hardware, which distinguishes them from traditional processors. In this way, operations are divided into pieces in FPGAs and multiple operations can be done simultaneously.

II. Image Processing Algorithms Used in the Study

Manuscript received January 20, 2015; revised March 25, 2015.

The authors are with Computer Engineering Department of Kocaeli University, Kocaeli, Turkey (e-mail: mehmetali.altuncu@kocaeli.edu.tr, tanerguven@gmail.com, ybecerikli@kocaeli.edu.tr, suhapsahin@kocaeli.edu.tr).

The RGB to grayscale conversion method is a commonly used method in image processing applications. The reason for the frequent use of this method instead of colorful images is to reduce the computational requirements of the operations that will be performed on the image. In this way, a higher performance is obtained in these applications [3]. There are several ways of converting a RGB image to a gray image. In this study, we converted the images to gray by using the formula given in (1). This method was designed in accordance with the compatibility of human eye with brightness perception by using a weighted combination of RGB channels. Moreover, this method is frequently used in computer vision and the "rgb2gray" function of MATLAB also uses this formula [4].

$$Y = 0.2989R + 0.5870G + 0.1140B \tag{1}$$

Convolution is another important algorithm commonly used in image processing. Convolution is an operation that executes the total of multiplication of image matrix values and kernel matrix values. Convolution is used to apply the image a spatial low and high pass filter. Depending on which kernel matrix is used in convolution, such operations as edge detection, sharpening and blurring can be performed on image [5], [6]. Although the dimensions of kernel matrixes vary depending on the application used, 3×3 matrixes are commonly used. Mathematically, 2-D convolution is represented using the following (2)

$$\begin{aligned} y(m,n) &= x(m,n) \times h(m,n) \\ &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} x[k,l]h[m-k,n-l] \end{aligned} \tag{2}$$

where x denotes input image, y denotes output image and h is the convolution kernel matrix [7].

III. System Design

A. The FPGA Card Used

The ZedBoard Zynq-7000 FPGA development kit [8] was used in this study. Zynq-7000 is different from standard FPGAs in that it contains Artix-7 FPGA and ARM Cortex-A9 processor on the same chip together. In a FPGA card having this system, those parts containing intense computations are performed on FPGA and the control parts not containing computations can be done on the processor by using software.

DOI: 10.7763/IJIEE.2015.V5.582

The internal structure of Zynq is shown in Fig. 1.

Zynq consists of two parts: Processing System and Programmable Logic. Processing System (PS) works like a traditional processor since it contains structures such as ARM Cortex-A9, Floating Point Unit, Memory Controller, Gigabit Ethernet Controller and USB Controller. Programmable Logic part, on the other hand, contains all the structures of a standard FPGA. The communication between PS and PL parts is provided by high performance data-paths.

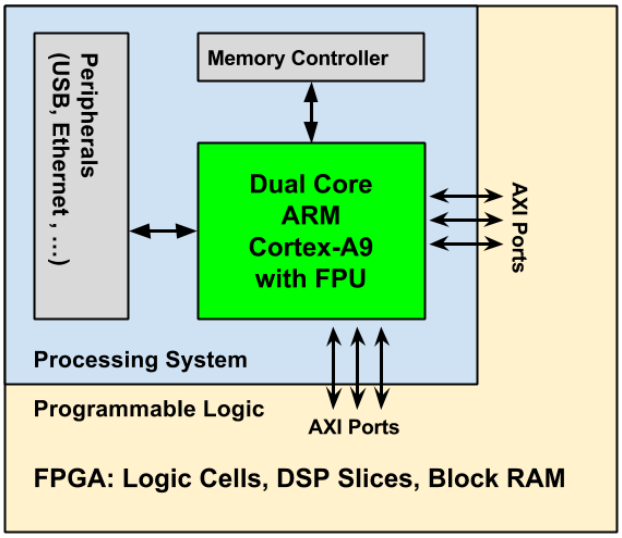


Fig. 1. The internal structure of Zynq.

B. General System

The developed system consists of two parts: hardware and software. The software performs the operation of image capturing from the video source and transferring it to the hardware part. Any image source that can run on Linux (such as USB camera, network video stream, video file etc.) can be used in the software. In the hardware part, image processing algorithms are implemented. The schema of the general system formed by the co-use of software and hardware parts is shown in Fig. 2.

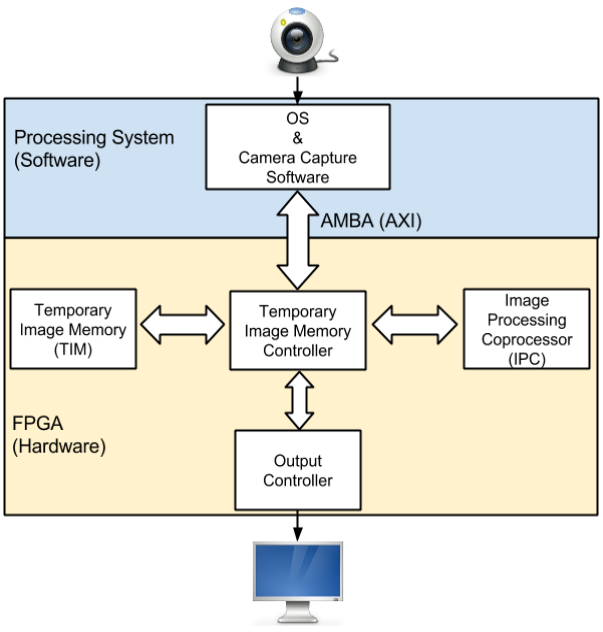


Fig. 2. The schema of the general system.

In the system, video frame is firstly obtained from the video source via software and operating system. Then, the software transfers the parts to be processed on the video frame to the temporary image memory on the FPGA. The transfer of operation, on the hardware part, is controlled by Temporary Image Memory Controller (TIMC) module. The video is transferred to Temporary Image Memory (TIM) module via the TIMC. The processed and unprocessed forms of the image can be stored in the TIM module.

Image Processing Co-processor (IPC) module performs image processing algorithms. The IPC module performs the image processing operations by fetching the pixels from TIM module and storing the processed forms of the pixels. These modules are explained in the 4th part. After the operations on the image are completed on the hardware part, the status notification is transmitted to the software. After this step, the processed image can be read via the software part if needed. The original and processed forms of the image are shown on the monitor by using the HDMI output of FPGA kit.

C. Hardware/Software Interface

In the study, Advanced Extensible Interface (AXI), which is an interface standard of Advanced Microcontroller Bus Architecture (AMBA) is used, for the connection between Processing System (PS) and Programmable Logic (PL). AXI is the whole of the standardized protocols which enable the data transfer among hardware modules [9]. AXI can work in two different ways as memory mapped and stream. Memory mapped AXI is used in this study. Memory mapped AXI works by associating the memory and registers in the module with memory addresses on the part of PS. The access to PL by PS is in the form of reading and writing operations on the associated address.

In the study, two different AXI connections, image transfer and control, were used for the communication between PS and PL. In the AXI connection used for image transfer, it is possible to access the whole of Temporary Image Memory. The register that shows the progress of processing is also available on the control connection.

The AXI connection used for image transfer was run at 32 bit mode. Image in 24 bit RGB format was obtained from the camera, but it was transferred to FPGA as 32 bit. The reason for the transfer of pixels as 32 bit is to ensure one pixel transfer at each clock cycle. In this way, the address transformations in the hardware part and the read and write operations in the memory were implemented more easily and by using less resource. The transformation from 24 bit to 32 was performed as the resetting the highest 8 bit. Thus, the need for additional operations on the pixels in the software part was eliminated.

D. Software

Video capturing from sources such as USB camera, network video stream and video file is a very complicated and difficult operation on FPGAs when compared to the software. Moreover, for video capturing, it is necessary to design special hardware for the video source. The aim of the system we developed is to perform real time operations on the video rather than video capturing itself. Therefore, using software for video capturing enabled us a great deal of convenience.

The ARM processor on the development kit we used can run an operating system. Linux operating system and userspace libraries provide a standard interface for the hardware. By this means, it is possible to access video sources easily via the software. Thus, software developed on the Linux operating system was used in video capturing part.

The software we developed captures images from the camera at certain intervals according to the frame rate used. After each pixel of the image to be processed is transferred to hardware part, the operation is expected to be completed by doing busy waiting with software. When the operation is complete, the hardware part notifies the software about the completion status of the operation with a flag. Then, the software passes to the next portion in the frame if it is available. After all the pixels in the frame were processed, the next frame is transferred from the camera and the same operations are repeated. Each frame was used as a single image portion in the test system.

This study was implemented by capturing images from USB camera via a software developed in C programming language by using a Video for Linux (v4l2) [10] library. Different image resources can also be used in the study by making changes only on the software and without any change in the hardware design. For example, network video stream or a video file on the memory card can be used as video source.

IV. Hardware Design

Image processing methods are performed by IPC module. TIM module consists of two memory units storing original images and the processed images. During the image processing operations, required pixels of the original image are read from TIM and processed pixels are written back. TIMC functions as a bridge during these operations. In Fig. 3, the symbolic diagram of the modules used in image processing and their contents are presented.

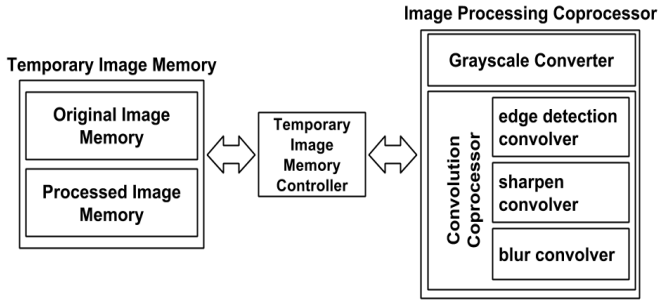


Fig. 3. The symbolic diagram of the modules.

A. Temporary Image Memory

In this module, two memory systems of 24 bit width and pixel number depth were used. One of the memories stores the original form of the image while the other one stores the processed image. Address ranges of both memories are between zero and pixel number (rows × columns). Memories are addressed as 24 bit format rather than byte format. As the used image format is 24 RGB, each memory cell was designed to capture one pixel.

Memories were implemented in the format of "Dual-Port RAM with Synchronous Read" [11]. There are two ports in the RAM in this system. One of the ports can only perform reading operations while the other can perform both reading and writing operations. The use of a RAM with two ports enabled us to performed video processing operations and display the images on the screen simultaneously.

B. Grayscale Converter

In the application, multiplication was done with coefficients close to the values in (1) in order to decrease the use of resources in grayscale conversion. The multiplication was done as bit shift and addition [12]. The formula we applied for grayscale conversion is shown in (3, 4, 5, 6).

$$\begin{aligned} r &= R \times 0.296875 \\ &= R \times (0.25 + 0.03125 + 0.015625) \\ &= R >> 2 + R >> 5 + R >> 6 \end{aligned} \tag{3}$$

$$\begin{aligned} g &= G \times 0.59375 \\ &= G \times (0.5 + 0.0625 + 0.03125) \\ &= G >> 1 + G >> 4 + G >> 5 \end{aligned} \tag{4}$$

$$\begin{aligned} b &= B \times 0.109375 \\ &= B \times (0.0625 + 0.03125 + 0.015625) \\ &= B >> 4 + B >> 5 + B >> 6 \end{aligned} \tag{5}$$

$$Y = r + g + b \tag{6}$$

where ">>" denotes right shift.

C. Convolution Coprocessor Design

Convolution co-processor operates on the 3x3 window and computes the value of pixel in the center. The computation is performed by multiplying the 9 pixels in the window with the values in the convolution kernel and the addition of the multiplication results. This total is taken as 255 if it is higher than 255 and it is taken 0 if it is smaller than 0. Since the convolution kernels used in the application contain primes of 2 or close coefficients, the multiplication operations are implemented as shifting and addition. The computation of a pixels value is presented with an example (7, 8).

$$H = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \tag{7}$$

$$\begin{aligned} y_{11} = (x_{00} + (x_{01} << 1) + x_{02} + \\ (x_{10} << 1) + (x_{11} << 2) + (x_{12} << 1) + \\ x_{20} + (x_{21} << 1) + x_{22}) >> 4 \end{aligned} \tag{8}$$

In the example shown in (8), multiplication with 2 is performed by 1 bit left shift and multiplication with 4 is performed by 2 bit left shift. To divide the results of the totals into 16, 4 bit right shift was applied.

The operation of scanning image pixels is performed as in the study of [13], by adding a new pixel to convolution buffer at each clock cycle and the elimination of the oldest pixel.

Two FIFO buffers (delay lines) were used to store the pixels between the rows in the 3x3 convolution window (see Fig. 4).

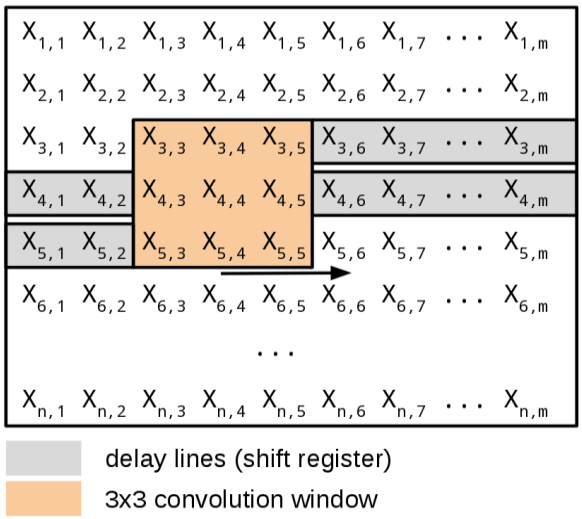


Fig. 4. Displacement of the 3×3 convolution window.

A long data-path is necessary for logic design since there are a great number of sequential operations in the computation of a pixel. The design implemented the entire operation at 1 clock cycle was tested in the development phase. It was seen that the adequate frequency couldn't be reached for real time running in that design. The data-path was shortened by the division of the operation into multiple pieces to enable the system to run at high frequency. By this way, the system could be run at higher frequencies. Since the divided operations were implemented by pipeline method, the entire image was computed in clock cycle equal to the pixel number. The block diagram of the convolver is presented in Fig. 5.

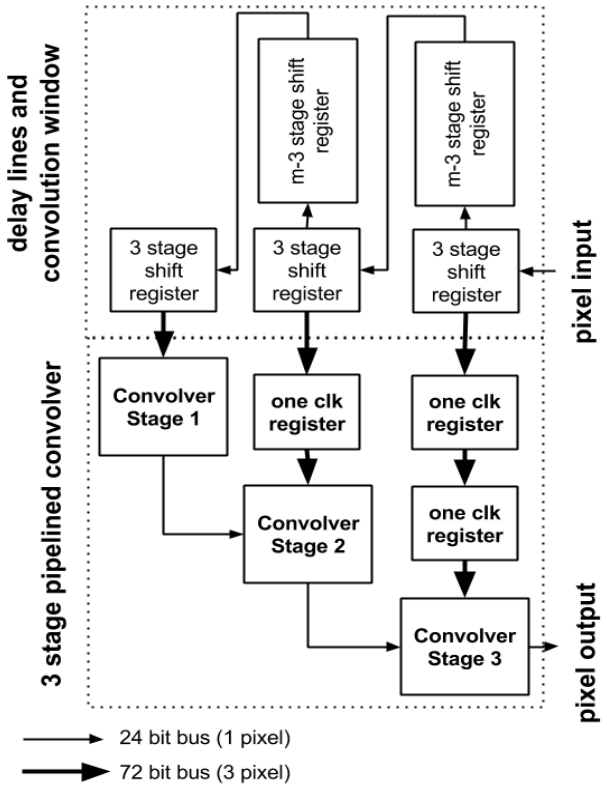


Fig. 5. The block diagram of the convolver.

V. Experimental Results

In Fig. 6, the photo of the test system taken during the study is presented. ZedBoard, HDMI monitor, standard USB camera and USB HUB were used in the test system.

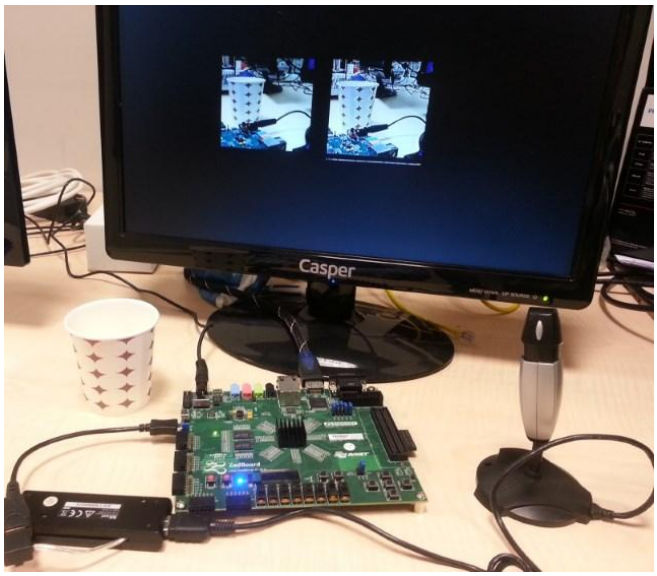


Fig. 6. The photo of the test system.

The design was implemented in the Xilinx ZedBoard Zynq-7020 FPGA by Vivado 2014.2. The device utilization summary is given in Table. I.

TABLE I: Utilization Summary

Resource	Utilization	Available	Utilization %
FF	3652	106400	3.43
LUT	2648	53200	4.98
Memory LUT	208	17400	1.20
I/O	46	200	23.00
BRAM	97.5	140	69.64
BUFG	5	32	15.62
PLL	1	4	25.00

Four different convolution kernels were implemented on the FPGA in the study. The convolution kernels used are shown in (9, 10, 11, 12). We used (9, 10) for edge detection, (11) for sharpening and (12) for blurring operations.

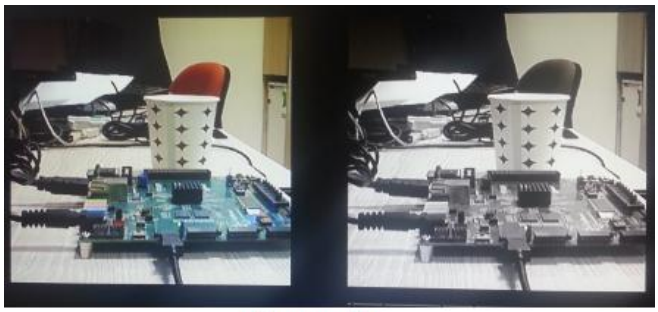
$$H = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{9}$$

$$H = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \tag{10}$$

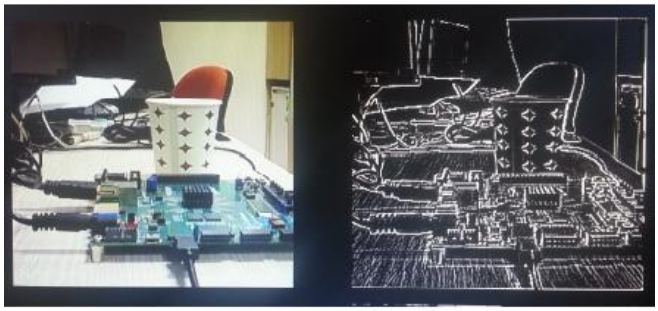
$$H = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} \tag{11}$$

$$H = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \quad (12)$$

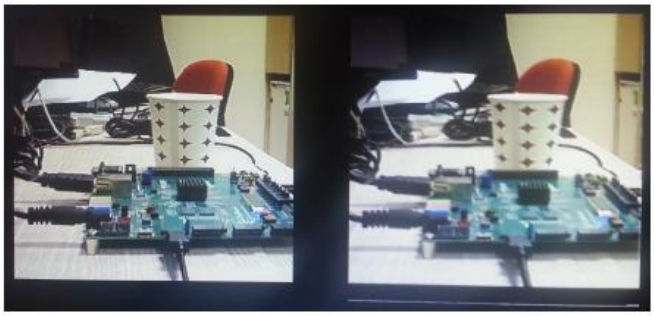
The convolution kernel being used can be changed during the study. In Fig. 7, screen prints of some method used are shown.



Grayscale



Edge Detection



Blurring

Fig. 7. Results.

The operations were performed on a 256×256 image and convolution coprocessor clock frequency was used as 150 Mhz in the developed system. The time spent for each frame is approximately 0.44 ms. Convolution coprocessor has a design that has the capacity to run at a higher speed than 2000 fps (frame per second). The transfer of frame takes roughly 25 ms with the implementation of the AXI we used. Therefore, approximately 40 fps speed could be reached in real time system.

VI. Conclusion

This paper presents the implementation of real time image processing algorithms with Hardware / Software Co-design on FPGA. At the end of the study, a real time working system of 256×256 image and approximately 40 fps were obtained. Although convolution coprocessor design is enough for working on large scale images, this study is limited with small scale images. The reason for that is the inadequacy of image transfer design to FPGA we developed. For further study, it is aimed to speed up the image transfer to FPGA and, by this means, work on high resolution images real time by the use of Direct Memory Access (DMA) implementation.

References

[1] M. A. Vega-Rodr guez, J. M. S ánchez-P érez, and J. A. G ómez-Pulido, "An FPGA-based implementation for median filter meeting real-time requirements of automated visual inspection systems", presented at 10th Mediterranean Conference on Control and Automation -- MED2002 Lisbon, Portugal, July 9-12, 2002.

[2] M. Kiran, K. M. War, L. M. Kuan, L. K. Meng, and L. W. Kin, "Implementing image processing algorithms using hardware in the loop approach for Xilinx FPGA" in *Proc. the International Confereence on Electronic Design*, pp. 1-6, December 1-3, 2008.

[3] C. Kanan and G. Cottrell, "Color-to-grayscale: Does the method matter in image recognition?" *PLoS ONE*, vol. 7, no. e29740, 2012.

[4] W. K Pratt, *Digital Image Processing PIKS Scientific Inside*, 4th Edition, New York: Wiley-Interscience, John Wiley & Sons, 2007.

[5] S. Jayaraman, S. Esakkirajan, and T. Veerakumar, *Digital Image Processing*, TMH, 2008.

[6] A. Ç. Bağbaba, B. Ors, and A. T. Erozan, "G örüntü iyileştirme işlemcisi ve uygulamalari," presented at Signal Processing and Communications Applications Conference, Trabzon, Turkey.

[7] A. E. Nelson, "Implementation of image processing algorithms on FPGA hardware," Master of Science Thesis, Faculty of the Graduate School of Vanderbilt University, Nashville, TN-USA, 2000.

[8] Xilinx. [Online]. Available: http://www.xilinx.com/support/university/boards-portfolio/xup-boards/XUPZedBoard.html

[9] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq Book Tutorials*, Glasgow: University of Strathclyde, 2014.

[10] A. Cox, Video4Linux programming, 2000.

[11] Xilinx Inc. (2008). XST User Guide 10.1. [Online]. Available: http://www.xilinx.com

[12] S. Kilts, *Advanced FPGA Design: Architecture, Implementation, and Optimization*, Wiley, 2007.

[13] G. Bois and Y. Savaria, "Reconfigurable pipelined 2D convolvers for fast digital signal processing," *IEEE VLSI Systems Transactions*, vol. 7, no. 3, September 1999.

Mehmet Ali Altuncu has received the B.S. degree in computer engineering from Sakarya University in 2010 and he is a M.Sc. student from Kocaeli University. Currently, he is a research assistant and master student in the Department of Computer Engineering and a researcher in Embedded Systems Laboratory at Kocaeli University, Kocaeli, Turkey. His research interest is in signal and image processing, embedded systems.

Taner Guven has received the B.S. degree in computer engineering from Kocaeli University in 2013. Currently, he is a master student in the Department of Computer Engineering and a researcher in Embedded Systems Laboratory at Kocaeli University, in Kocaeli, Turkey.

Yasar Becerikli has received the B.S. degree in electronics and communication engineering from Yildiz Technical University in 1991 and got his M.Sc in 1994 from Istanbul Technical University and his PhD degrees in 1998 from Sakarya University. He is currently working as a professor in Kocaeli University, Kocaeli, Turkey. His research interest is in signal and image processing, control theory, fuzzy systems and neural networks.

Suhap Sahin has received the PhD degree in electronics and communication engineering from Kocaeli University in 2009. He is currently working as an assistant professor and a director of Embedded Systems Laboratory in Computer Engineering at Kocaeli University, Kocaeli, Turkey. His research interest is in computer architecture, logic design, mobile system design and embedded systems.