Acceleration of image processing algorithms using minimal resources of custom reconfigurable hardware

John V. Vourvoulakis *
John Lygouras
Section of Electronics & Information Systems Technology
Department of Electrical & Computer Engineering,
Democritus University of Thrace,
Polytechnic school of Xanthi, Greece
* Corresponding author, e-mail address: jvourv@ee.duth.gr

John A. Kalomiros
Technological and Educational Institute of Serres,
Department of Informatics and Communications,
Terma Magnisias, 62100 Serres, Greece

Abstract—**The hardware/software implementation of a custom vision board using minimal resources out of a reconfigurable platform is described. Demanding robotic vision applications in most cases require dedicated hardware for reliable operation. The designed system is based on a Cyclone IV Altera FPGA device that constitutes the main processing unit of the reconfigurable hardware and on a 32–bit Microchip PIC32 microcontroller as a complementary processor. The main goal of this research is to implement image processing algorithms using only minimal resources of the FPGA device. The microcontroller serves peripheral control tasks, relieving valuable resources from FPGA. Video images are captured using a CMOS image sensor. USB connectivity with a personal computer is provided using a FIFO-to-USB module. Operational tasks such as frame grabbing, image filtering and USB communication are integrated to the system by implementing custom-designed controllers in VHDL. Image processing functions are accelerated using a fully parallel pipeline which is described analytically. A host computer interface has also been developed in order to test the overall system in action. The system is evaluated in terms of resource usage and the advantages emanating from the proposed architecture are discussed.**

Keywords; FPGA; parallelism; processing algorithm acceleration; reconfigurable hardware

I. INTRODUCTION

During the last decades robotic applications were tremendously increased. Automotive industry, electronics manufacturing and aerospace are certain fields in which robotic vision is applied. Demanding robotic vision capabilities require dedicated hardware since it is more reliable and faster than software solutions. Evolution of FPGA technology has proved the use of reconfigurable hardware very attractive [1] and as a result more researchers choose FPGAs to host their designs [2-4]. Parallel processing is the basis for accelerating iterative algorithms that are comprised of complex computations, like convolution, Fourier Transforms or other DSP operations. The more complex the computations, the more resources out of the FPGA device are needed. Although FPGA manufacturers are continuously announcing more powerful devices, resource usage constitutes an important issue for every such system.

In this paper we present a methodology to implement image processing algorithms and various input-output peripherals hosted at a custom low-cost circuit board appropriate for machine vision. Every incorporated task is based on custom controllers designed in VHDL. For mere synthesis and configuration, the standard Altera Quartus II software platform is used. We avoid the use of more sophisticated tools for system-on-a-chip design, like Qsys, which may shorten design-cycle but on the other hand are often heavily dependent on commercial controllers and intellectual property (IP) cores. Also bypassing the use of general purpose IP controllers we reduce needed resources to the lowest possible level, since our custom controllers include only the necessary features. A well designed and optimized custom controller in most cases requires fewer resources in comparison to general purpose multi-featured controllers.

The board is based on Altera's Cyclone IV family FPGAs and is minimally equipped with peripheral devices, allowing a large number of pins and chip resources to be used for acquisition and processing tasks. Implemented input-output peripherals include a frame-grabber from a CMOS image sensor, a USB controller for host communication all designed in VHDL. Additional peripheral functionality and complementary processing is supported using a Microchip 32-bit PIC microcontroller. As a consequence of the adopted design concept, the total system's cost is maintained very low. The expenditure for a special purpose commercial development board, dedicated to video processing, can rise to hundrends or even a few thousands of euros, which is much more than the final cost of the proposed custom system.

Following from the above consideration, the main contribution of this paper is twofold. On the one hand the development of a reconfigurable platform capable of hosting advanced image processing and control applications, with the lowest possible resources, is described. The system is based on custom controllers for frame grabbing, USB or VGA communication and also allocates adequate resources for a minimal on-chip RAM memory. On the other hand, the total cost of the system is kept at a very low level. The choice of a low cost FPGA device, the small cost of the microcontroller chip and the avoidance of purchasing expensive copyrighted IP cores give the opportunity to limit research funding only to the mandatory.

The rest of the paper is organized as follows. In Section II the system's hardware architecture is presented and some details about the system components are provided. In Sections III and IV the image acquisition stage and the asynchronous communication with FIFO-to-USB module are analyzed. In Section V a parallel implementation of several trivial image processing tasks, namely edge detection, mean value and Gauss image filters is presented. Section VI presents experimental results and an evaluation of the system in terms of resource usage and frame rates. Advantages of the system architecture and future work are also discussed. Section VII concludes the paper.

II. System Architecture

The system hardware follows a modular architecture. This means that every capability incorporated in the system requires an appropriate custom hardware interconnection module. The block diagram of the system is depicted in Fig. 1.

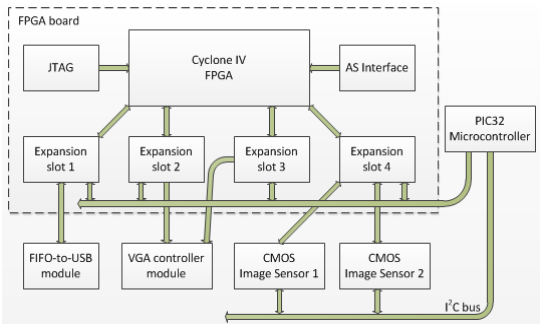

Figure 1. The block diagram of the hardware architecture.

The main processing unit is consisted of a Cyclone IV EP4CE22E22C7 Altera FPGA device [5], its necessary external components such as voltage regulators, decoupling capacitors, a 50 MHz crystal oscillator and two header connectors for device configuration. The first header connector is for JTAG interface and is mainly used as long as development is in progress. The second one is for FPGA configuration with serial configuration memory (Active Serial interface) and can be used when development has been finalized. The above two headers have been designed for interconnection with USB Blaster Download Cable. Moreover all I/O FPGA pins are connected to four additional header connectors that can be considered as board expansion slots.

The PIC32 microcontroller has also four header connectors used as expansion slots for I/Os. It is currently connected to the I^2C interface of the CMOS image sensors. The communication between the FPGA device and the microcontroller is attained using free I/Os from both chips. If there is no need for exchanging information they can be disconnected from each other in order to preserve I/O resources.

For capturing image data the 5 Mpixel MT9P031 CMOS color image sensor from Aptina Imaging [6] on the MT9P031I12STCH header board was used. The header board consists of the MT9P031, suitable lens and all the external components the image sensor needs in order to be functional. The sensor has a parallel digital interface for transmitting data and a serial I^2C interface for configuration. Internal ADC has 12-bit resolution, providing 4096 color scales. In our system we use the 8 most significant bits from the ADC, since they are adequate for our current research purposes. Interconnection with the FPGA device includes data signals (D3 to D11) and control signals, frame valid (FV), line valid (LV) and pixel clock (PIXCLK).

USB connectivity with a host computer is obtained using the UM232H FIFO-to-USB module from FTDIChip [7]. It is based on FT232H chip and provides USB2.0 high speed connectivity. Control and bulk transfers according to USB protocol are hardwired. All necessary descriptor information for the enumeration procedure is saved on external EEPROM by the manufacturer at production time. The associated used signals are eight data bits and four control signals TXE, RXF, WR and RD. TXE shows if USB module can accept transmit data and RXF shows if there are available data to be read. When signals WR and RD are active, write and read internal sequences are launched respectively.

Real-time image processing applications require a rate of many frames per second. It is well known that in real-time applications computers may fail due to a huge load of concurrent processes. Equipping the board with VGA connectivity can be a good solution to this issue. A custom hardware module was designed and implemented, comprised of the ADV1723 high speed triple video DAC from Analog Devices [8]. There are three 8-bit data buses that pass color information to the VGA module. Timing synchronization is obtained using control signals HS and VS.

Interconnection between FPGA, microcontroller and peripheral custom-designed modules is depicted in Fig. 2.

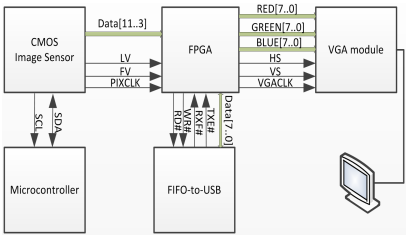

Figure 2. Processing units and peripheral modules connectivity.

III. The Image Grabber

The principal prerequisite for every task we implement is frame grabbing. Every other functionality receives image frames as input. Before analyzing the VHDL implementation details we first examine how the CMOS image sensor produces data.

MT9P031 outputs color component information of image pixels in a progressive scan. Pixel data start from top right corner of the first row and end up to the bottom left corner of the last row. Intervals between consecutive rows are referred as horizontal blanking and between consecutive frames as vertical blanking. Control signals FV and LV declare when sensor outputs data. When FV and LV signals are noticed '1' then sensor launches pixel data on every rising edge of PIXCLK. Output data are considered to be valid and can be read from

FPGA on the next falling edge of PIXCLK. MT9P031 outputs image data using Bayer encoding. Bayer encoding describes every pixel by reducing color information to one byte instead of three. Even rows use the pattern Green-Red-Green-Red and odd rows use the pattern Blue-Green-Blue-Green. RGB color information for every pixel is extracted from its neighbors. The pattern usually is referred to as Color Filter Array (CFA) and the procedure of extracting full color information is called demosaicing.

The Frame Grabber is implemented in VHDL using state machines. The clock used for the state machines is PIXCLK and is derived from the image sensor. The flow of state machines is depicted in Fig. 3.

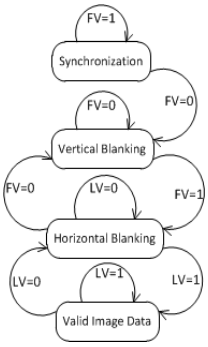

Figure 3. Image readout state machines.

In the “Synchronization” state the system just waits. When FV signal is asserted '0' then this declares that the sensor is in the vertical blanking interval, which means that the FPGA is now synchronized and can proceed to the “VerticalBlanking” state. When FV is asserted '1' then the FPGA enters into “HorizontalBlanking” state and if LV is also asserted '1' then it enters into “ValidImageData” state. Now, the device can read sensor's output. When the sensor completes transmission of the first row's pixel data then asserts LV to '0' and the FPGA device enters into “HorizontalBlanking” state. When horizontal blanking interval is over, LV is asserted again to '1' and the FPGA enters back to the “ValidImageData” state. This sequence will continue until sensor completes transmission of the last image row. Afterwards LV and FV are asserted '0' consecutively and the FPGA enters first into “HorizontalBlanking” state and finally into “VerticalBlanking” state.

Image data are stored in on-chip memory. We allocate 16Kbytes of Cyclone's internal memory to store data. The on-chip RAM is not enough to store entire frames. This is not a problem since our system supports full parallelism for internal operations. Stored data are processed (if required) and sent to the output device before RAM becomes insufficient. When the buffer is full then subsequent data overwrite prior data. Parallelism provides satisfactory functionality even if we do not have external RAM at our disposal.

IV. USB COMMUNICATION

The system uses the UM232H FIFO-to-USB module for USB connectivity. It takes over all low and high level operations for a bidirectional communication with computer's USB port. It uses a First In First Out buffer for transmitted or received data and also has a specific communication interface for write and read purposes. At this point of our research we use the USB module in asynchronous operation mode. This mode does not need a clock to exchange data. The timing diagram for a typical write is shown in Fig. 4.

The FPGA device must assert interface signals according to write or read sequence in order to transmit or receive data. Apart from the correct order, signal assertion is subject to timing constraints. Timing constraints are defined from manufacturer and for successful transactions must be adhered precisely.

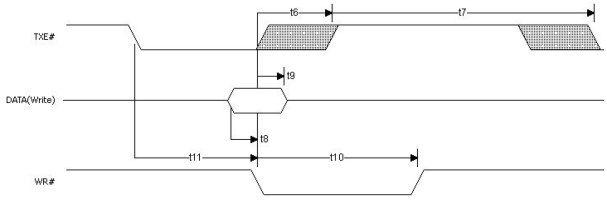

Figure 4. Asynchronous FIFO interface – WRITE signal waveforms.

A suitable controller has been implemented in VHDL using state machines. The clock used for this controller is 50MHz and is derived from the external crystal oscillator. The state machines are optimized for this execution speed. The write sequence of state machines is depicted in Fig. 5.

At the beginning, the FPGA controller stays at “Idle” state. In this state no data are sent. CMD constitutes an internal control signal which is asserted to '1' when FPGA wants to transmit data. When the CMD signal is asserted '1' from a process, then the controller enters to “Send” state. It stays in that state for as long as the TXE# signal is asserted '1', meaning that UM232H is busy or the FIFO buffer is full. When TXE# is noticed '0' the controller asserts the WR signal to '0', launches data on the data bus and enters into state “Intermediate 1”. Transition to state “Intermediate 2” occurs on the next clock pulse and finally the procedure arrives to the “Complete state”. It waits there until the CMD signal is asserted '0' by the internal process that asserted it '1' and then it returns to “Idle” state. The overall state machines sequence is designed to be fully compatible with UM232H write sequence.

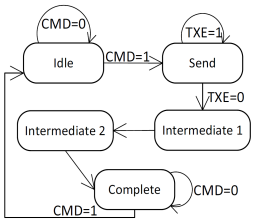

Figure 5. WRITE sequence State Machines.

From the host computer side, an appropriate software application has been developed in Visual Basic. This application receives image data and displays them on the screen. It uses D2XX vendor's driver for USB communication. The host application checks if there are incoming data every

time a timer expires, which is approximately every 10 ms. If there are available data, the application reads them and reconstructs the image. When image reconstruction is completed, the software shows it up in a picture-box component. When a new frame is received the previous loaded image in the picture-box is refreshed. Depending on an external stimulus applied to our board, the host application can display the output result from different task-logic blocks. As discussed in the following paragraph, an example is a number of different image filters that are implemented simultaneously in the FPGA device.

V. IMAGE FILTERS IMPLEMENTATION

Parallelism not only accelerates operations but provides flexibility in hardware architecture. In our implementation parallelism of processing algorithms is performed inside "ValidImageData" state. We have developed certain simple tasks such as a Mean filter, a Gauss filter and an Edge Detector. The aforementioned tasks are usually required when we study advanced topics like stereo processing, feature extraction or object recognition.

These three filters operate simultaneously in our VHDL implementation. Below, the 3x3 kernel matrices applied on input image are quoted. Kernel M is for mean filter, G is for Gauss filter and P_1, P_2 are Prewitt masks for Edge Detector.

$$M=\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix},\quad G=\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

$$P_1=\begin{bmatrix}1 & 1 & 1\\ 0 & 0 & 0\\ -1 & -1 & -1\end{bmatrix},\quad P_2=\begin{bmatrix}-1 & 0 & 1\\ -1 & 0 & 1\\ -1 & 0 & 1\end{bmatrix}$$

Parallelism requires the pixel intensities of the 3x3 image area, where the convolution kernel is applied, to be simultaneously available. As we have already mentioned, sensor outputs color component information using Bayer encoding. The total procedure requires two intermediate steps in order to complete processing. The first step is to extract pixel intensities from Bayer encoded data for every 3x3 image window and the second step is to apply the filter mask. The hardware structure is presented on Fig. 6.

Parallelism is achieved using RAM-based shift registers. Sensor data are pipelined into shift registers. They are designed in a specific way which outputs image data from 3x3 image window in a progressive scan. There are two blocks of shift registers. Every block has two RAM-based shift registers. They also consisted of six single shift registers synthesized by logic elements. The functions "Demosaicing/Intensity Extraction" and "Mean Filter/Gauss Filter/Prewitt Mask" are asynchronous operations and propagation delay of the signals determine maximum processing speed. The available image window for both shift register blocks is depicted in Fig. 7.

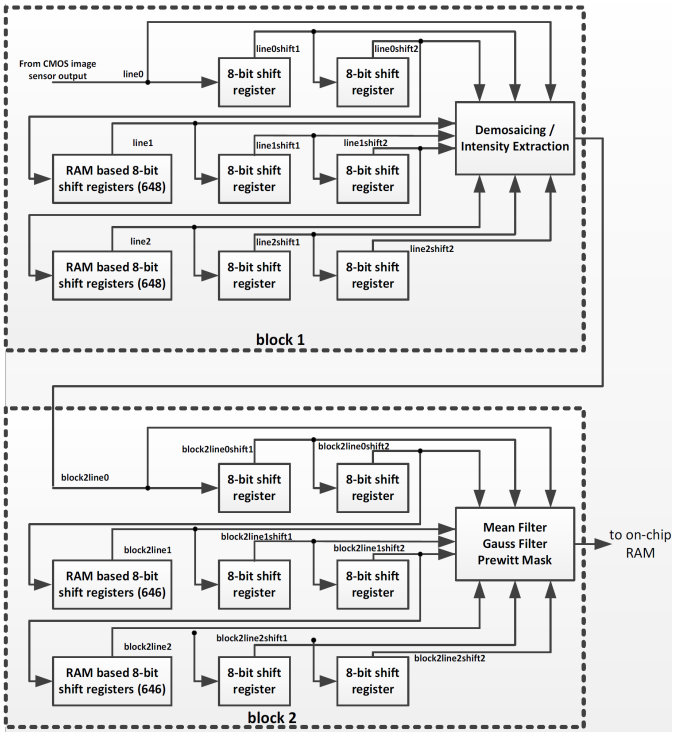

Figure 6. Hardware structure in FPGA for parallelism.

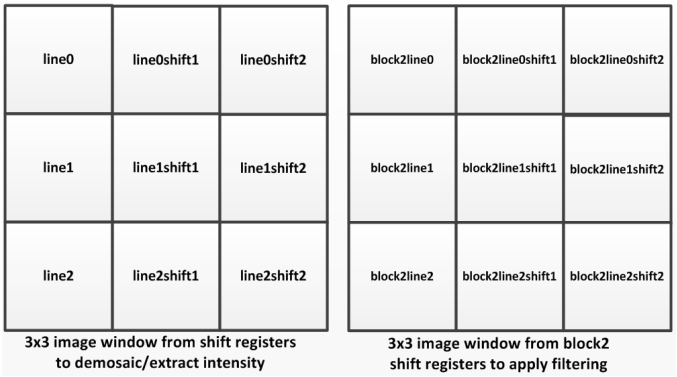

Figure 7. Available image window for parallel operations.

Demosaicing is followed by convolution using the image masks, while image pixels are streaming through shift registers. The results of the convolution are stored in on-chip RAM memory. The function Demosaicing can be implemented using several ways. In the literature many algorithms have been proposed [9]. One very simple algorithm is to use the mean value of neighboring colors. Depending on the known color component information of a pixel they are named as Gr, Gb, R or B. Table II presents how full color information is calculated in every case. After the Demosaicing procedure completes calculations for a pixel that belongs to row i and column j, intensities are extracted using adding the three color component information and dividing by 3.

TABLE I. DEMOSAICING PIXEL DATA

PIXEL	Full Color Component calculation

Gr	Red = (R(*j-1,i*) + R(*j+1,i*))/2
	Green = Gr
	Blue = (B(*j,i-1*) + B(*j,i+1*))/2
Gb	Red = (R(*j,i-1*) + R(*j,i+1*))/2
	Green = Gb
	Blue = (B(*j-1,i*) + B(*j+1,i*))/2
R	Red = R
	G1 = Gr(*j-1,i*) + Gr(*j+1,i*) G1 = Gb(*j,i-1*) + Gb(*j,i-1*) Green = (G1 + G2)/4
	B1 = B(*j-1,i-1*) + B(*j-1,i+1*) B2 = B(*j+1,i+1*) + B(*j+1,i-1*) Blue = (B1 + B2)/4
B	R1 = R(*j-1,i-1*) + R(*j-1,i+1*) R2 = R(*j+1,i-1*) + R(*j+1,i+1*) Red = (R1 + R2)/4
	G1 = Gb(*j-1,i*) + Gb(*j+1,i*) G2 = Gr(*j,i-1*) + Gr(*j,i+1*) Green = (G1 + G2)/4
	Blue = B

From now on we follow the convention of naming a pixel that is at i^{th} row and j^{th} column as $p_{j,i}$. Let us consider the example of applying aforementioned filters on a 3x3 image window to describe how this mechanism works in detail. A random window of input image is illustrated in Fig. 8. We may focus on the window that is highlighted with bold borders. This is a 4x4 window. Assume that the bottom right pixel of that window belongs to the random row *i* and column *j* of the input image. In order to apply a filter mask at $p_{j-2,i-2}$ we need to know pixel intensities from a 3x3 window the bottom right pixel of which is $p_{j-1,i-1}$. This means that the processing elements must have already calculated the intensity of $p_{j-1,i-1}$ and have it available for use. This premises that demosaicing of $p_{j-1,i-1}$ has been completed. In order to demosaic $p_{j-1,i-1}$ and then extract the intensity of that pixel we need to know the color component from the 3x3 window of which the bottom right pixel is $p_{j,i}$. This implies that only when FGPA has readout $p_{j,i}$ it will be able to perform filtering computations on $p_{j-2,i-2}$. Hence FPGA must have available pixel information that are being in the bold 4x4 window to be able to apply the processing algorithm on pixel $p_{j-2,i-2}$.

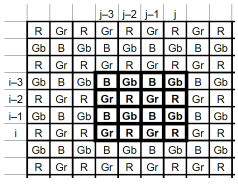

Figure 8. Image window necessary for 3x3 convolutions.

Mean filter implementation for pixel $p_{j-2,i-2}$ includes first the calculation of the following matrix.

$$M_{p_{j-2,i-2}} = \begin{bmatrix} 1 \cdot I_{j-3,i-3} & 1 \cdot I_{j-2,i-3} & 1 \cdot I_{j-1,i-3} \\ 1 \cdot I_{j-3,i-2} & 1 \cdot I_{j-2,i-2} & 1 \cdot I_{j-1,i-2} \\ 1 \cdot I_{j-3,i-1} & 1 \cdot I_{j-2,i-1} & 1 \cdot I_{j-1,i-1} \end{bmatrix}$$

Naming every element of matrix M as $m_{x,y}$, where x=1,2,3 and y=1,2,3 the output image is produced as in (1).

$$I'_{j-2,i-2} = \frac{1}{9} \cdot \sum_{x=1}^{3} \sum_{y=1}^{3} m_{y,x} \quad (1)$$

Gauss filter implementation for pixel $p_{j-2,i-2}$ includes the calculation of the following matrix.

$$G_{p_{j-2,i-2}} = \begin{bmatrix} 1 \cdot I_{j-3,i-3} & 2 \cdot I_{j-2,i-3} & 1 \cdot I_{j-1,i-3} \\ 2 \cdot I_{j-3,i-2} & 4 \cdot I_{j-2,i-2} & 2 \cdot I_{j-1,i-2} \\ 1 \cdot I_{j-3,i-1} & 2 \cdot I_{j-2,i-1} & 1 \cdot I_{j-1,i-1} \end{bmatrix}$$

Naming every element of matrix G as $g_{x,y}$, where x=1,2,3 and y=1,2,3 the output image is produced as in (2).

$$I'_{j-2,i-2} = \frac{1}{16} \cdot \sum_{x=1}^{3} \sum_{y=1}^{3} g_{y,x} \quad (2)$$

Edge Detector implementation is slightly different from Mean and Gauss filters. First two Prewitt matrices P_1 and P_2 are calculated for the pixel $p_{j-2,i-2}$.

$$P_{(1)p_{j-2,i-2}} = \begin{bmatrix} -1 \cdot I_{j-3,i-3} & 0 \cdot I_{j-2,i-3} & 1 \cdot I_{j-1,i-3} \\ -1 \cdot I_{j-3,i-2} & 0 \cdot I_{j-2,i-2} & 1 \cdot I_{j-1,i-2} \\ -1 \cdot I_{j-3,i-1} & 0 \cdot I_{j-2,i-1} & 1 \cdot I_{j-1,i-1} \end{bmatrix}$$

$$P_{(2)p_{j-2,i-2}} = \begin{bmatrix} 1 \cdot I_{j-3,i-3} & 1 \cdot I_{j-2,i-3} & 1 \cdot I_{j-1,i-3} \\ 0 \cdot I_{j-3,i-2} & 0 \cdot I_{j-2,i-2} & 0 \cdot I_{j-1,i-2} \\ -1 \cdot I_{j-3,i-1} & -1 \cdot I_{j-2,i-1} & -1 \cdot I_{j-1,i-1} \end{bmatrix}$$

Considering every element of matrix P_1 and P_2 as $p_{(1)x,y}$ and $p_{(2)x,y}$ where x=1,2,3 and y=1,2,3 the output image is produced as in (3).

The magnitude of image gradient is produced as the sum of the absolute values of horizontal and vertical gradients instead of the square root of the sum of the squares of matrix elements. It is simpler, produces similar results and requires less hardware resources. In order to receive a binary edge image a thresholding procedure is applied. The gradient threshold is defined as T=40.

$$I'_{j-2,i-2}=\begin{cases}0, & \sum_{x=1}^{3}\sum_{y=1}^{3} p_{(1)x,y} + \sum_{x=1}^{3}\sum_{y=1}^{3} p_{(2)x,y} < T \\ 255, & \sum_{x=1}^{3}\sum_{y=1}^{3} p_{(1)x,y} + \sum_{x=1}^{3}\sum_{y=1}^{3} p_{(2)x,y} > T\end{cases} \quad (3)$$

VI. Experimental Results and System Evaluation

In this section we present image results that are produced setting the overall system in action. All images have resolution 640x480. In Fig. 9 test results are presented from the Frame Grabber and the Edge Detector.

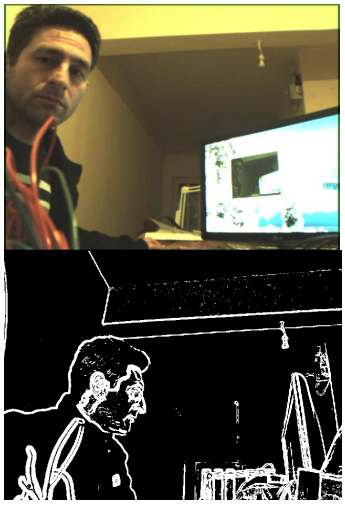

Figure 9. Image results from Frame and Edge Detector.

Evaluation of a system is a function of many aspects. The proposed custom system for video processing was basically motivated by the need of reduced overall cost in conjunction with open source code and flexible architecture. Overall performance is also an important issue. The final prototype board is demonstrated in Fig. 11 and is actually a compromise between the above objectives.

Table III gives the necessary hardware resources for the implementation of the task logic presented in the previous sections. In order to implement simultaneously a mean filter, a Gauss filter and a Prewitt edge detector we used less than 10% of the available logic elements and almost half the available on-chip memory. This is needed for a frame size 640x480 while for smaller images with half VGA resolution, the necessary memory bits are almost 75000. The reported low resource usage gives us the opportunity to add more features and task logic to our system.

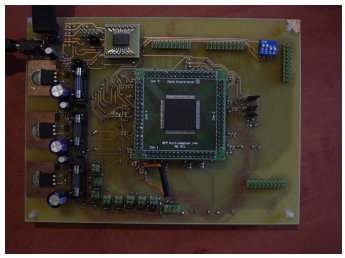

Figure 10. The prototype FPGA board.

TABLE II. Resources needed for frame resolution 640x480

Recources	Available	Used	Percentage
Logic elements	22320	2000	9,00%
Pins (I/O)	80	42	52,50%
Memory bits	608256	290904	47,83%

The implemented FPGA task logic is capable of processing 162 frames per second in full VGA resolution or 650 half VGA frames per second (fps), with a crystal oscillator at 50MHz. Practical processing frame rate is much more limited. The image sensor can capture at most 53 fps in VGA resolution. With its present USB configuration, the overall system in action sends to the computer 4 full VGA frames per second, or 16 half VGA frames. In fact, the host application controlling USB connectivity in asynchronous USB mode, slows down the communication procedure. Using a more powerful computer or developing more sophisticated software can lead to a better overall performance.

VII. Conclusions

A parallel architecture of image processing functions applied on a low-cost custom video-processing FPGA/microcontroller board, is presented. The system features a Cyclone IV Altera device implementing the required video processing task and a PIC32 microcontroller able to support peripheral functions. All processing functions and custom controllers are developed in VHDL reserving a minimal of available resources. The video board features a frame grabber suitable for CMOS image sensors using Bayer encoding and a USB module, providing connectivity to a host computer for further processing. The cost of the proposed board is very low compared to existing commercial video kits. The system is expandable and is indented to host demanding machine vision applications.

References

[1] U. Meyer_Baese, “Digital Signal Processing with Field Programmable Gate Arrays,” Springer Berlin, Heidelberg, New York, 2007.

[2] W.J. MacLean, “An evaluation of the suitability of FPGAs for embedded vision systems,” in: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05), vol. 3, San Diego, California, USA, June 2005, p. 131.

[3] T. Cervero, S. Lopez, G.M. Callico, F. Tobajas, V. de Armas, J. Lopez, R. Sarmiento, “Survey of reconfigurable architectures for multimedia applications,” VLSI Circuits and Systems IV, edited by T. Riesgo, E. de la Torre, L. S. Indrusiak, Proc. of SPIE Vol. 7363, 736303, 2009, pp. 1-11.

[4] J.A. Kalomiros, J. Lygouras, “Design and evaluation of a hardware/software FPGA-based system for fast image processing,” Microprocessors and Microsystems, Volume 32, Issue 2, March 2008, Pages 95-106.

[5] Altera Home-page: <http://www.altera.com/>.

[6] Aptina Imaging Home-page: <http://www.aptina.com/>.

[7] FTDIChip Home-page: <http://www.ftdichip.com/>.

[8] Analog Devices Home-page: <http://www.analog.com/>.

[9] Daniele Menon, Giancarlo Calvagn, “Color image demosaicking: An overview,” Signal Processing: Image Communication, Volume 26, Issue 8-9, October 2011, Pages 518-533.

[10] I.S. Uzun, A. Amira, A. Bouridane, “FPGA implementations of fast Fourier transforms for real time signal and image processing,” IEE Proceedings—Vision, Image and Signal Processing 152 (3) (2005) 283–296.