Design and Implementation of a novel FPGA – based Image Acquisition System with two CMOS Sensors for Advanced Processing Techniques

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 novel image acquisition system, designed to host advanced processing techniques, is introduced. The system is suitable for robotic vision applications, such as stereo image processing, visual odometry etc. The architecture is based on a Cyclone IV Altera FPGA device that constitutes the main processing unit and on a 32–bit Microchip PIC32 microcontroller as a complementary processor. The microcontroller undertakes peripheral control tasks, relieving valuable resources from the FPGA. The system can capture image data simultaneously from two CMOS sensors and store necessary image rows in FPGA's on chip memory. Moreover, it uses a FIFO-to-USB module to transfer plain or processed image data to a host computer. The system also supports VGA connectivity. Operational tasks such as frame grabbing, image processing and communication with high-speed USB module are implemented in VHDL. A host computer interface has also been developed in order to test the overall system in action. The system is evaluated in terms of real-time performance and the advantages emanating from the proposed architecture are discussed.**

Keywords; FPGA; CMOS image sensor; USB connectivity; image processing; VHDL; hardware design

I. INTRODUCTION

Over the last decades image processing techniques evolved rapidly providing continuously better results on various tasks. On the other hand they demand increasingly more resources from computational systems. Parallel processing is the basis for accelerating iterative algorithms that are comprised of complex computations, like convolution, Fourier Transforms or other DSP operations. Evolution of FPGA technology with its ability for run-time reconfiguration has opened new horizons to embedded programmers and designers [1]. As a result more researchers choose FPGAs to host their designs [2-4].

The target platform and the development software tools for every potential project are always two main design issues. FPGA manufacturers and also third party vendors have produced numerous development boards with various specifications. New versions of software development tools are often released by manufacturers in order to support new devices and to fix bugs in older versions.

In this paper we propose a custom low-cost circuit board appropriate for real-time machine vision and control tasks. 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 and a VGA controller. Frame-grabbing and host-communication functions were among the first to be developed for our custom hardware board since they are necessary for testing, debugging, monitoring and demonstrating purposes. An on-chip dual-clock RAM memory has also been included in the main architecture. Additional peripheral functionality and complementary processing is supported using a Microchip 32-bit PIC microcontroller. Relieving the FPGA device from an overhead of fixed additional controllers, like ADC and DAC converters or SDRAM external memory interfaces, gives a flexibility to dedicate more resources on the requested task. 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.

The basic input-output and processing stages of the proposed system-on-a-chip are custom-designed in VHDL, which is standard for research and industry. 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 IP cores. In this way, system development is maintained unaffected from software updates and compatibility issues that may arise between software versions. Also, the cost overhead associated with the purchase of copyrighted intellectual property (IP) is avoided.

Following from the above considerations, 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. An on-board microcontroller expands the processing capabilities and peripheral functionality of the board allowing for hardware/software co-design. 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 details about the system components are provided. The inter-connectivity between system stages is analytically explained. In Sections III and IV the image acquisition stage and the asynchronous communication with FIFO-to-USB module are analyzed in detail. In Section V an implementation of several trivial image processing tasks, namely edge detection, mean value and Gauss image filters is presented, as a demonstration of the functionality of the proposed board. Section VI presents experimental results from the aforementioned functions. An evaluation of the system is presented in Section VII, in terms of resource usage and frame rates. Advantages of the system architecture and future work are also discussed. Section VIII concludes the paper.

II. Hardware Architecture

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

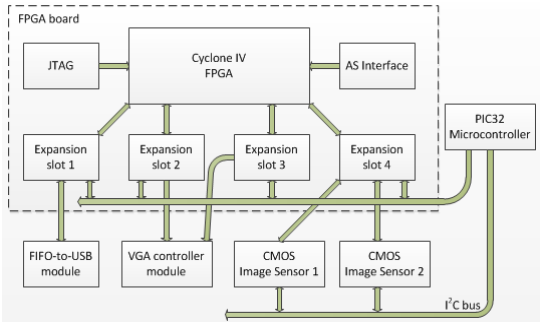

Figure 1. The block diagram of the hardware architecture.

The main processing unit is consisted of a Cyclone IV EP4CE22E22C7 Altera FPGA device, its necessary external components such as voltage regulators, decoupling capacitors, crystal oscillator and two header connectors for device configuration. Voltage regulators produces three voltage levels from the main power source. FPGA core needs 1.2V for proper operation, internal PLL supply circuits demand 2.5V and the third voltage was specified at 3.3V for use with FPGA's I/O interface. A 50 MHz crystal oscillator provides the main clock for FPGA's internal synchronous operations. 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 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). The interconnection between image sensor, FPGA device and microcontroller is depicted in Fig. 2.

Figure 2. Interconnection FPGA-image sensor-microcontroller.

A desirable and useful capability for every image acquisition, processing and control system is transmitting processing results to a personal computer. In many cases they can be used for further analysis, evaluation or as visual input for computer-based robotic algorithms. In order to incorporate this feature to our system we used the UM232H FIFO-to-USB module from FTDIChip. It is based on FT232H chip and provides USB2.0 high speed connectivity supporting various operation modes. 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. At this phase of our research, the USB module is configured for asynchronous operation which supports up to 8 Mbytes/s transfer rate. In this operation mode the associated signals are eight data bits and four control signals TXE#, RXF#, WR# and RD#. The sharp mark on the right of every signal declares that they are active low. 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. In Fig. 3 the interconnection between the USB module and the FPGA device is presented.

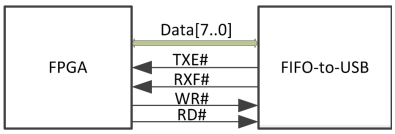

Figure 3. Interconnection between FPGA and FIFO-to-USB.

Real-time image processing applications require a rate of many frames per second. Demonstration and testing purposes often demand fast image displaying on a screen. Even a fast connection like USB in conjunction with a very fast personal computer may fail to respond timely to a task like this. 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. In order to support the aforementioned capability a custom hardware module was designed and implemented, comprised of the ADV1723 high speed video DAC from Analog Devices. In Fig. 4 the interconnection between the custom module and the FPGA device is depicted. There are three 8-bit data buses that pass color information to the VGA module. Timing synchronization (which is also referred as Horizontal and Vertical Synchronization) is obtained using control signals HS and VS. The VGA clock depends on the resolution we choose to display or on the supported screen resolution.

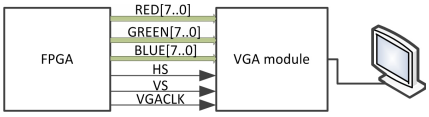

Figure 4. Interconnection between FPGA and VGA module.

III. The Image Grabber

The principal prerequisite for every task we implement is frame grabbing. Every other functionality receives image frames as input. The Frame Grabber has been developed in VHDL but before analyzing the 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. When image is fully read then we can extract RGB color information for every pixel 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 readout order is presented in Fig. 5 and the timing diagram of an image readout is depicted in Fig. 6.

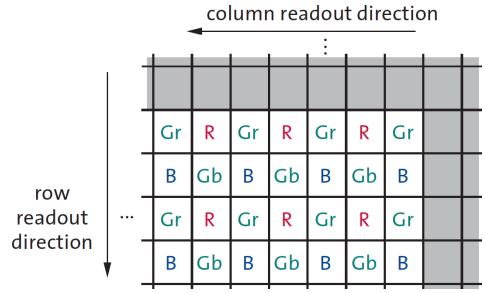

Figure 5. Readout order of Bayer encoding.

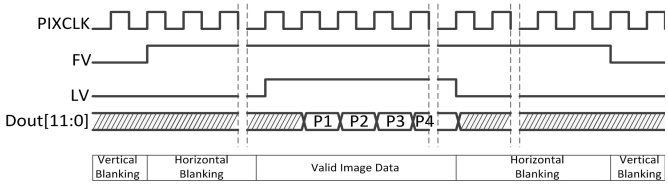

Figure 6. Timing diagram of an image readout.

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. 7.

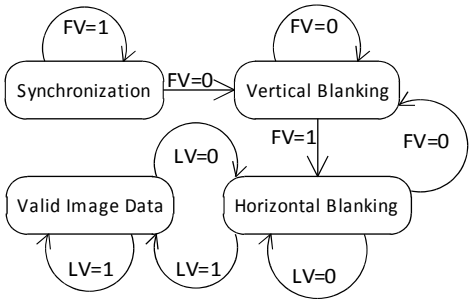

Figure 7. Image readout state machines.

In the “Synchronization” state the system just waits. This state has been added to enforce the FPGA device to wait until the next frame generation in case that power was applied to the FPGA device while the sensor was already streaming image data at an intermediate point of a frame. 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, as sensor data are considered valid at every falling edge of PIXCLK. 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. The FPGA device remains there until the sensor starts sending the next frame.

Image data are stored in on-chip memory. We allocate 32Kbytes of Cyclone's internal memory to store data coming from two image sensors (16Kbytes for each one). The on-chip RAM is not enough to store entire frames. As a consequence when the buffer is full then subsequent data overwrite prior data. This is not a problem since our system supports full parallelism of processing operations. This means that stored data are processed and sent to the output device before RAM becomes insufficient. Generally the following sequence takes place inside the FPGA device:

- Read pixel data and save to the temporary register block (to be used for potential processing purposes)
- Read next pixel data and save to the temporary register block – previous data saved to the temporary register block are stored to RAM at location 1
- Read next pixel data and save to the temporary register block – previous data saved to the temporary register block are stored to RAM at location 2 – data stored at location 1 are sent to the output device
- Read next pixel data and save to the temporary register block – previous data saved to the temporary register block are stored to RAM at location 3 – data stored at location 2 are sent to the output device

When all memory locations are full of data then the above sequence starts again from the first location overwriting previous records. Parallelism provides satisfactory functionality even if we do not have external RAM memory at our disposal.

IV. USB Communication

It is often the case that processed image results are needed to be available to a personal computer. They can be input to a computer-based algorithm for further processing or just need to be stored for further evaluation. By transmitting results to a personal computer, debugging purposes can be served as well.

This capability is incorporated to our system providing USB connectivity with a computer. The system uses the UM232H FIFO-to-USB module. 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. 8 and for a typical read is presented in Fig. 9.

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. Signal timing constraints related to read and write procedures are quoted in Table I.

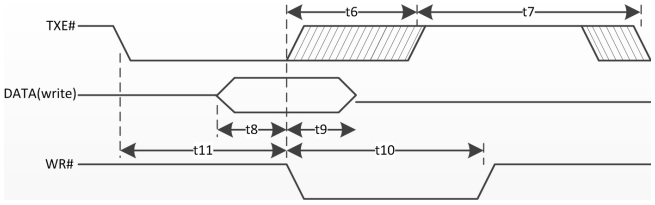

Figure 8. Asynchronous FIFO interface – WRITE signal waveforms.

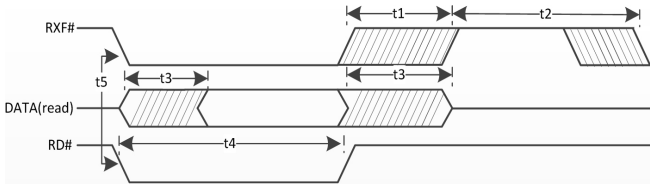

Figure 9. Asynchronous FIFO interface – READ signal waveforms.

TABLE I. Timing Constraints of UM232H WRITE and READ sequences

Time	Description	Min	Max	Units
T1	RD# inactive to RXF#	1	14	ns
T2	RXF# inactive after RD# cycle	49		ns
T3	RD# to DATA	1	14	ns
T4	RD# active pulse width	30		ns
T5	RD# active after RXF#	0		ns
t6	WR# active to TXE# inactive	1	14	ns
t7	TXE# active to TXE# after WR# cycle	49		ns
t8	DATA to WR# active setup time	5		ns
t9	DATA hold time after WR# inactive	5		ns
t10	WR# active pulse width	30		ns
t11	WR# active after TXE#	0		ns

In our application we concentrate on the write interface. 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 is depicted in Fig. 10.

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.

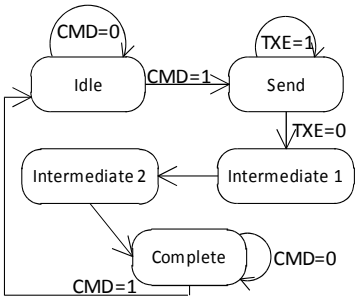

Figure 10. WRITE sequence State Machines.

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 basic idea is that 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. Different stimuli are obtained by using three push-button switches.

V. Implementation of Image Filters

As we have discussed in previous sections parallelism not only accelerates operations but provides flexibility in hardware architecture. In our implementation parallelism of processing algorithms is performed inside “ValidImageData” state. Parallel operations can start when necessary data are pipelined inside the FPGA device. The target function specifies the size of pipelined data according to its requirements. In order to test our system and evaluate the performance 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. Depending on external stimulus our system forwards the result of the selected filter to the output. 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.

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

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

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. 11.

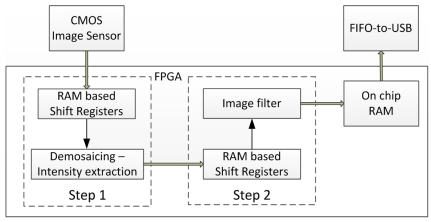

Figure 11. Hardware structure in FPGA for parallelism.

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 subwindows in a progressive scan. Assuming that shift registers are full of data the pipelining procedure is described by the following steps:

- Sensor outputs pixel data to shift registers in step 1 and demosaics/extracts intensity from the central pixel that belongs to image's subwindow 1
- Sensor outputs next pixel data to shift registers in step 1 and demosaics/extracts intensity from the central pixel that belongs to image's subwindow 2 – intensity pixel data are pipelined into shift registers in step 2 and image filter applied on image's subwindow 1
- Sensor outputs next pixel data to shift registers in step 1 and demosaics/extracts intensity from the central pixel that belongs to image's subwindow 3 – intensity pixel data are pipelined into shift registers in step 2 and image filter applied on image's subwindow 2 – filter results on subwindow 1 are saved to RAM
- Sensor outputs next pixel data to shift registers in step 1 and demosaics/extracts intensity from the central pixel that belongs to image's subwindow 4 – intensity pixel data are pipelined into shift registers in step 2 and image filter applied on image's subwindow 3 – filter results on subwindow 2 are saved to RAM

The above procedure is being executed continuously until image subwindows scan entire image.

The function Demosaicing in step 1 can be implemented using several ways. In literature many algorithms have been proposed [6]. 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, as in Fig. 5. 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 (1).

$$I(j,i)=(Red+Green+Blue)/3 \qquad (1)$$

TABLE II. 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	$Red=R$
	$G_1=Gr(j-1,i)+Gr(j+1,i)$ $G_2=Gb(j,i-1)+Gb(j,i+1)$ $Green=(G_1+G_2)/4$
	$B_1=B(j-1,i-1)+B(j-1,i+1)$ $B_2=B(j+1,i+1)+B(j+1,i-1)$ $Blue=(B_1+B_2)/4$
B	$R_1=R(j-1,i-1)+R(j-1,i+1)$ $R_2=R(j+1,i-1)+R(j+1,i+1)$ $Red=(R_1+R_2)/4$
	$G_1=Gb(j-1,i)+Gb(j+1,i)$ $G_2=Gr(j,i-1)+Gr(j,i+1)$ $Green=(G_1+G_2)/4$
	$Blue=B$

Intensity calculations and filtering are carried out concurrently while FPGA is reading subsequent pixel data. When FPGA reads data from row *i* it also completes processing algorithms on previous rows. 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. 12. Let us 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 $p_{j-2,i-2}$.

				j–3	j–2	j–1	j		
	R	Gr	R	Gr	R	Gr	R	Gr	R
	Gb	B	Gb	B	Gb	B	Gb	B	Gb
	R	Gr	R	Gr	R	Gr	R	Gr	R
i–3	Gb	B	Gb	**B**	**Gb**	**B**	**Gb**	B	Gb
i–2	R	Gr	R	**Gr**	**R**	**Gr**	**R**	Gr	R
i–1	Gb	B	Gb	**B**	**Gb**	**B**	**Gb**	B	Gb
i	R	Gr	R	**Gr**	**R**	**Gr**	**R**	Gr	R
	Gb	B	Gb	B	Gb	B	Gb	B	Gb
	R	Gr	R	Gr	R	Gr	R	Gr	R

Figure 12. 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 (2).

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

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 (3).

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

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 (4).

$$I'_{j-2,i-2}=\begin{cases}0, & \left|\sum_{x=1}^{3}\sum_{y=1}^{3}p_{(1)x,y}\right|+\left|\sum_{x=1}^{3}\sum_{y=1}^{3}p_{(2)x,y}\right|<T\\ 255, & \left|\sum_{x=1}^{3}\sum_{y=1}^{3}p_{(1)x,y}\right|+\left|\sum_{x=1}^{3}\sum_{y=1}^{3}p_{(2)x,y}\right|>T\end{cases} \quad (4)$$

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.

VI. Experimental Results

In this section we present image results that are produced setting the overall system in action. All images have resolution 640x480. In Fig. 13 test results are presented from the Frame Grabber. There are two images. The first image is in Bayer encoding as produced by the sensor and displayed on the computer screen without any further processing. The second image is derived by demosaicing the first image. Let us note that extraction of full color information has been placed on the computer software in order to reduce transmit load from the FPGA device to the computer. Fig. 14 shows results from the Edge Detector. The first image is the output of the Frame Grabber in gray scale. The second is the image produced applying the Prewitt masks and the thresholding procedure.

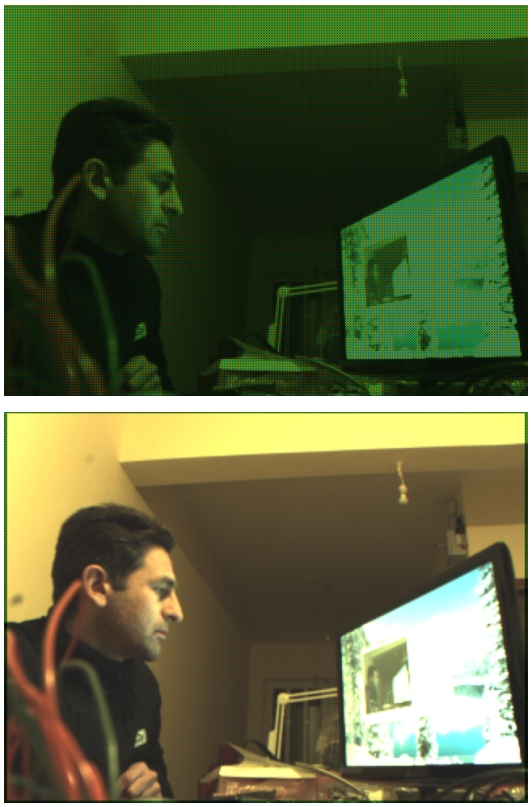

Figure 13. Image results from the Frame Grabber.

Figure 14. Image results from the Edge Detector.

VII. System Evaluation

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 flexible architecture. Overall performance is also an important issue. The final prototype board is demonstrated in Fig. 15 and is actually a compromise between the above objectives. The total cost of the system is about 60 euros, including the microcontroller, FIFO-to-USB module, the Cyclone IV FPGA device and the serial configuration device. We consider this total cost to be very satisfactory.

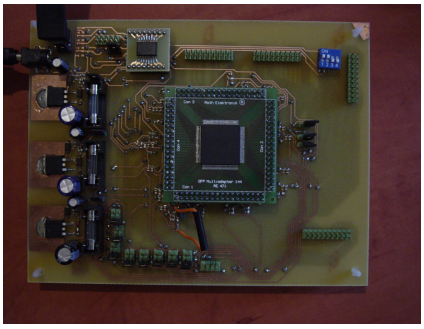

Figure 15. The prototype FPGA board.

The proposed board is a minimal but powerful video processor that can employ only the essential system resources, while at the same time it can expand in a modular and flexible manner in order to comply with additional requirements. Part of the power and flexibility is found in the proposed combination of a reconfigurable device with a well known, powerful 32-bit processor. A harware/software data and control interface is under development for the seamless connection between the processor and the reconfigurable hardware.

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.

Let us note that the VHDL implementation for a VGA controller is still in progress and therefore the required resources are not taken into account in Table III. The main limitation associated with VGA is relevant to the available I/O pins. However, the flexibility of the proposed architecture allows excluding other supplementary capabilities in order to host additional features. A VGA port can be added by simply removing USB connectivity, since the output information can now be displayed on the VGA screen.

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. This frame rate can be considered adequate for testing and debugging purposes or for evaluating and analyzing image results.

TABLE III. 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%

VIII. Conclusions

A low-cost video-processing custom FPGA/microcontroller board, based on a flexible modular architecture is presented. The system features a Cyclone IV Altera device implementing some video processing task and a PIC32 microcontroller able to support peripheral functions. 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 system is still in experimental design phase and it is proved to be able to host task logic for basic image processing using only a fraction of the available resources. 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 and real-time control 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] Hou, H., Zhang, W., Huang, D., Zhang, T., “Design and realization of real-time image acquisition and display system based on FPGA”, 2011 International Conference on Mechanical Engineering and Technology, ICMET 2011, London, November 24-25.

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

[7] 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.