computing@computingonline.net
www.computingonline.net

Print ISSN 1727-6209
On-line ISSN 2312-5381
International Journal of Computing

DESIGN DETAILS OF A LOW COST AND HIGH PERFORMANCE ROBOTIC VISION ARCHITECTURE

John V. Vourvoulakis [1], John A. Kalomiros [2] and John N. Lygouras [1]

[1] Democritus University of Thrace, Section of Electronics and Information Systems Technology, Department of Electrical and Computer Engineering, Polytechnic School of Xanthi, Xanthi, Greece, 67100, jvourv@ee.duth.gr
[2] Technological and Educational Institute of Central Macedonia, Department of Informatics Engineering, Terma Magnisias, Serres, Greece, 62124

Abstract: The implementation of an advanced real-time, low cost video processing platform capable of supporting a variety of demanding robotic applications is presented. The system is designed as an open project, accessible in full detail and has the potential to grow. It is based on a FPGA plus MCU architecture, allowing the implementation of combined fixed-point and 32-bit floating-point applications with optimized resource allocation. The presented platform is optimally integrated with appropriate controllers, like video-input frame grabbers for multiple camera applications, external SDRAM, as well as USB and VGA interfaces. The processing and interfacing capabilities of the proposed system are illustrated by implementing basic feature extraction and preprocessing tasks, achieving the display of processed video frames at a rate of 30 fps with resolution 640x480. The proposed architecture is evaluated in terms of resource usage, power consumption and cost. Potential applications are also discussed.

Keywords: FPGA, microcontroller, real-time systems, CMOS image sensor, image processing, reconfigurable hardware

1. INTRODUCTION

Over the last decades great progress has been made in the field of image and video processing. Areas of development include robotic vision, medical imaging, security monitoring, video games, satellite photography and etc. Robotic vision [1] is widely used in industrial manufacturing, assembly and inspection of components as well as in autonomous navigation of land, underwater and aerial vehicles. Image and video processing algorithms are computationally demanding, as they apply transformations on a per frame basis, often processing all different color planes [2,3]. It is well known that software techniques based on sequential structures often fail to achieve the performance desired for real-time applications, even when using powerful microcomputer systems. Numerous schemes and systems have been proposed to overcome this problem. One approach uses vision-specific software on conventional desktop platforms, often coupled with graphics accelerators [4,5]. Another approach uses dedicated embedded hardware [6-9]. The use of Field Programmable Gate Arrays (FPGAs) is an attractive solution to the implementation of vision-specific robotic applications. The main advantages of FPGA technology are parallelism of iterative algorithms, hardware reuse, low power consumption, design flexibility and lower cost in comparison with Application Specific Integrated Circuits (ASICs).

In this paper, we propose a custom low cost and scalable circuit board that is suitable for real-time machine vision and control tasks. The architecture is based on Altera's Cyclone IV family FPGA (EP4CE22E22C7 device) and is minimally equipped with peripheral devices, allowing a large number of pins and chip resources to be used for image acquisition and processing tasks. Additional peripheral functionality and complementary processing is supported by using a Microchip 32-bit PIC microcontroller (PIC32MX795F512H device). The microcontroller undertakes peripheral control tasks, relieving the FPGA device from an overhead of fixed additional controllers, like ADCs/DACs or other serial interface controllers. Also, it expands the processing capabilities and peripheral functionality of the board for future tasks.

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. Implemented input-output peripherals

include frame-grabber cores interfaced to CMOS image sensors, a high speed FIFO-to-USB controller module for host communication, a Video Graphics Array (VGA) controller for displaying image data on a computer screen, an optimized SDR SDRAM controller and a master Serial Peripheral Interface (SPI) core for serial communication with the microcontroller. The Altera Quartus II software platform is used for synthesis and configuration. We avoid the use of more sophisticated tools for system-on-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, the overhead cost associated with the purchase of copyrighted intellectual property (IP) is avoided.

Microcontroller firmware is developed in MPLAB X IDE. The program is written in C programming language and the XC32 compiler is used to translate the source code into machine code. The I^2C peripheral interface is initialized to configure appropriately the CMOS image sensors. Moreover, the use of SPI peripheral in conjunction with Direct Memory Access module achieves communication between the PIC and FPGA device without the intervention of the microcontroller's CPU.

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 hundreds or even a few thousands of euros, which is much more than the final cost of the proposed custom system.

Beside its low cost, the system is designed as an open robotic vision project with a potential to grow. The proposed architecture can be recreated and can be expanded in the future by adding hardware parts in the open HDL vision library introduced in this article.

Following from the above considerations, the main contribution of this paper is twofold. First, the development of a high performance reconfigurable platform, capable of hosting advanced image processing and control tasks, is described in reproducible detail. Also, the system is introduced as a project open to further development. Researchers can adopt design and build their applications taking advantage of the presented software and hardware functionality. Second, the total cost of the system is kept at a low level, due to key engineering choices and the use of optimized custom controllers and vision cores. Introducing a low cost, open-source vision platform achieving high performance is important, because the evolution of inexpensive machine vision processors advances other fields, such as educational and domestic robotics, exploration robots and surveillance systems as well as research in all the above fields.

The rest of the article is organized as follows. In Section 2, a literature survey is given. In Section 3, the system's hardware components and their connectivity are presented in detail. Section 4 refers to the system architecture, providing design diagrams for all available controllers and task modules in the HDL vision library. Brief descriptions of microcontroller firmware and host computer software are given in Section 5. In Section 6, the system is evaluated in terms of performance, resource usage, power consumption and cost. In Section 7, experimental results are presented. Section 8 concludes the paper and discusses issues on reproducing the system.

2. LITERATURE SURVEY

The literature on image processing algorithms and systems is vast. Image processing algorithms, methods and their capabilities are often surveyed by review papers, however there is a major shortage of such reviews for image processing hardware architectures. Therefore a comparative study of published work is essential in order to locate scientific and technical aspects or even trends that need to evolve.

Since the main processing unit of the proposed system is an FPGA device we emphasize on FPGA-based implementations. We can classify related work into two categories. The first category describes the design of image acquisition systems that also perform some simple pre-processing tasks. The point of interest is the realization of a complete architecture and its capabilities. Usually some trivial processing operations exemplify performance. In the second category, researchers focus on the implementation details of specific image processing algorithms and their parallelization inside the FPGA data path. Special ready-made commercial boards are mostly used to host the architecture. Benchmark images are tested in order to compare results between papers. The point of interest is not the overall system but the advances in the implementation of the applied algorithm. According to the above classification our work is placed in the first category, but it is aimed to be used for applications of the second category, with special emphasis given to maintaining the overall system's cost very low.

In Refs. [10-12], FPGA-based image acquisition systems interfacing with CMOS image sensors are presented. References [10] and [11] present design aspects and image results, however resource usage is not provided and processing rates are not discussed. They both use a mid-range cost development board to host the architecture. In [12], an image acquisition

system that achieves 25 fps using Camera Link interface is presented. Image resolution is 1280x1024 pixels. However, the authors do not provide implementation details, resource usage or image processing results. In [13], an image acquisition and remote transmission system is described. It is based on a low cost FPGA chip achieving almost 15 fps for its maximum resolution (2048x1536 pixels). It captures data from a CMOS image sensor and it also implements Ethernet transceiver, VGA DAC, SRAM and flash memory. Since a small FPGA is used, it would be interesting to know the resource requirements, however they are not provided.

The following papers go beyond image acquisition and perform some additional processing, applying certain algorithms. In [14], the presented System on a Programmable Chip (SoC) includes a Sobel Edge Detector. A description of the architecture is given, but image results, resource usage and frame rates are not provided. In [15] an active vision sensor is presented and a tracking algorithm is implemented. Authors apply a processing algorithm on a Window Of Interest (WOI) and they use an expensive Stratix FPGA to realize the architecture. An interesting low cost system is demonstrated in [16], utilizing a Digital Signal Controller (DSC) as the processing unit for an Edge Detector. The system fails to achieve real-time performance and is not able to host more demanding algorithms. A Sobel Edge Detector is also demonstrated in [17] achieving processing speed of 60 fps, for image resolution 720x480. This architecture is hosted on a high cost DSP development kit. In [18], a very fast and low cost Sobel Edge Detector is presented, but it is not clear if the reported speed is derived from experimental measurements or from timing analysis results. The same task is demonstrated in [19], where color information is used. The processing rate is 50 fps for 720x576 size images. The system used in this paper is the ML510 Virtex-5 FX130T, which is considered a high cost solution. Reference [20] shows an Edge Detector on Hexagonal Sample Image Grids. The proposed architecture stores frames in FPGA's internal memory and as a result it can be used only with very small images. In [21], an image pre-processing system is presented that uses an Auto White Balancing (AWB) algorithm to improve quality, producing more realistic colors. It claims a frame rate of 75 fps for 1280x1024 image resolution, using a costly FPGA chip from Xilinx Virtex-5 family.

In the following references, implementations of more advanced processing algorithms are explored. In [22], a high cost video development kit is used to acquire infrared images and accomplish image fusion tasks. A disparity map computation based on the Sum of Absolute Differences (SAD) algorithm is presented in [23-25]. A costly DSP Development Kit is used in [23], succeeding to produce 23 disparity maps for image resolution 640x480, with a maximum disparity range of 64 pixels. In [24], an Edge Detector produces binary images to apply SAD while dense disparity maps are produced by interpolation. The architecture is hosted in Xilinx ML505 Evaluation Platform and achieves a processing rate of 50 fps, for image resolution 1280x1024 pixels. A lower cost approach is presented in [25] with the novelty of an injective consistency check adopted for disparities validation. A Xilinx Virtex-4 XC4VLX60 FPGA chip is used and the supported frame rate is up to 97 fps. Although this implementation is less expensive in comparison with other systems, only the cost of the chip is about a few hundred euros. Another stereo vision system is presented in [26] applying census transform to solve the correspondence problem. It is based on a Xilinx Virtex-4 FPGA and generates 60 fps for image resolution of 640x480 pixels. An optical flow FPGA design is described in [27] using Altera's DE2-70 development board. The system receives images from a computer via RS-232 link and performs Horn and Schunk's method achieving the computation of the optical flow vector field for images of resolution 256x256 pixels in 3.89ms. Reference [28] combines an FPGA chip and a DSP in order to implement the original SIFT algorithm. The system detects SIFT features and extracts keypoint descriptors in real time, with computations per feature consuming 80 us. In [29], an embedded FPGA architecture is proposed for the computation of grey-level co-occurence matrices (GLCM) and Haralick’s texture features, based on small 128x128 image blocks.

The common disadvantage of the above implementations is that, in general, they make use of expensive FPGA devices and development boards. As a result, the potential total system's cost can rise to hundreds or even a few thousands of euros. The system proposed in this paper is designed as a complete standalone vision processing system. Special effort has been given to develop measures that prove the claimed functionality and performance. The cost of the prototype board is less than 80 euros, excluding the cost of the CMOS image sensors. Utilizing inexpensive CMOS image sensors, such as the Toshiba TCM8230MD can help to maintain the total cost of the video processing system below 100 euros, while the image sensor supports 30 fps for image resolution 640x480 pixels. If higher frame rates or resolutions are required, then a more expensive sensor should be used. In our implementation a 5 Mpixel Aptina Imaging sensor

was selected to interface with the system, as described in the following section.

One disadvantage of the proposed system is its limited resources in terms of available look-up tables and registers. The optimal design of the developed cores ensures minimal usage of resources by the dedicated interfaces, but even so several resource demanding computational tasks would be difficult to implement with the EP4CE22E22C7 FPGA device. However, review papers, such as [30] for stereo vision, aid to evaluate a system's suitability for a given algorithm.

3. SYSTEM HARDWARE ARCHITECTURE

The system is designed to be scalable. It has expansion slots where external hardware modules can be connected. Each one among the supported functionalities demands a hardware interfacing module and a proper controller core inside the FPGA. The block diagram of the system is depicted in Fig. 1.

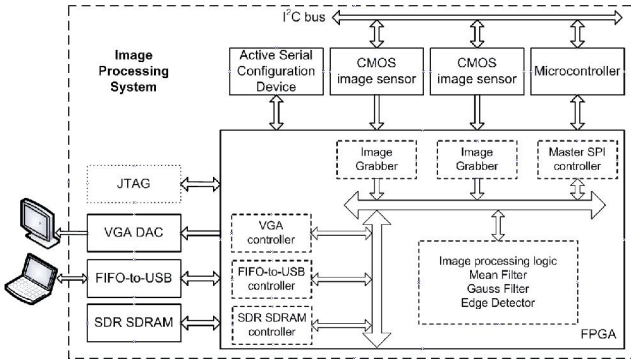


Fig. 1 – Block diagram of the system.

The main processing unit consists of an Altera Cyclone IV EP4CE22E22C7 FPGA device. It has 22K Logic Elements (LE), 80 available I/O pins, 594 Kbits internal SRAM, 132 9-bit multipliers and 4 PLLs. External components are placed on board to compose the necessary power supply voltages. A crystal oscillator and a JTAG header connector for device configuration are also incorporated on board. The FPGA core needs 1.2V for proper operation, internal PLL supply circuits demand 2.5V and for the communication with external devices a voltage level of 3.3V is required. A 50 MHz crystal oscillator provides the main clock for FPGA's synchronous operations. We could have selected a higher frequency for faster internal processing, but it would increase the power consumption and also the radiation emitted from the board. In the future, an appropriate oscillator replacement will be considered, depending on the target application. The JTAG interface is provided to configure the FPGA for as long as development is in progress. When the design is finalized the Active Serial Configuration Device (EPCS16N serial memory) is programmed through the FPGA using a JTAG Indirect Configuration file. Then, the EPCS16N device configures the FPGA every time the power supply is applied.

The complementary processing unit includes a Microchip 32-bit PIC32MX795F512H micro-controller unit (MCU). Communication with the FPGA device is attained through Serial Peripheral Interface. Transactions from SPI to microcontroller's memory and vice versa are accomplished using DMA transfers releasing the CPU to perform other processing tasks. If the supported target application does not need data exchange with the microcontroller unit, the MCU and the FPGA can be disconnected from each other in order to preserve I/O resources of both devices. The MCU's Inter-Integrated Circuit (I^2C) is used to configure the CMOS image sensors. Frame resolution, capture speed, shutter width, blanking intervals are some of the settings defined by the microcontroller.

Input images are captured from the 5 Mpixel MT9P031 CMOS color image sensor from Aptina Imaging on the MT9P031I12STCH header board. The header board consists of the MT9P031 device, a suitable lens and other external components needed by the image sensor. The sensor supports a capture speed of 14 frames per second (fps) at its full resolution of 2592x1944 pixels. For VGA resolution (640x480 pixels) it supports a maximum of 123 fps. It provides a parallel digital interface for transmitting data and a serial I^2C interface for configuration. The Analog to Digital Converter (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 (D4 to D11) and control signals, frame valid (FV), line valid (LV) and pixel clock (PIXCLK). The interconnection between the image sensors, the FPGA device and the microcontroller is depicted in Fig. 2.

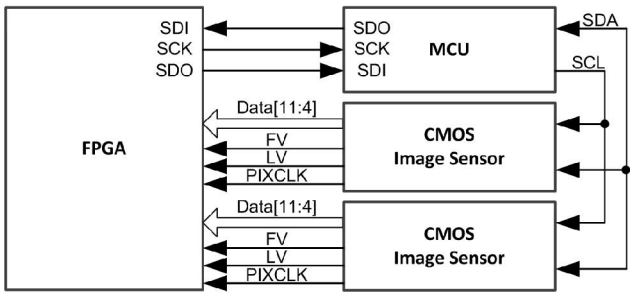


Fig. 2 – Interconnection between the CMOS image sensors, the microcontroller and the FPGA device.

A desirable and useful capability for every image acquisition, processing and control system is the transmission of processing results to a personal computer. In our target applications, the transmitted results can be used as a visual input for computer-based robotic algorithms or for evaluation. 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 bits $\overline{\text{TXE}}$, $\overline{\text{RXF}}$, $\overline{\text{WR}}$ and $\overline{\text{RD}}$. These signals are active low. The interconnection between FPGA and UM232H is depicted in Fig. 3.

Real-time image processing requires 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 due to a huge load of concurrent processes. Equipping the board with VGA connectivity is a good solution to this issue. In order to support the aforementioned capability, a custom hardware module was designed and implemented, comprised of the ADV7123 high speed video DAC from Analog Devices. Although the chip ADV7123 supports three 10-bit inputs for every color component, we use one in order to preserve I/O FPGA pins and as a result only gray-scale images can be displayed on the screen. Timing synchronization, which is also referred to as Horizontal and Vertical Synchronization, is obtained using control signals HS and VS. The VGA clock depends on the selected screen resolution.

In Fig. 3 the interconnection scheme between the FPGA device and SDRAM, VGA DAC and FIFO-to-USB module is illustrated. At this point of system development, either VGA connectivity or USB connectivity is available, due to limited I/O FPGA pins. Nevertheless, in most cases, this is not a restriction. Supporting both features simultaneously is rather redundant, since it is necessary only for a few specific applications.

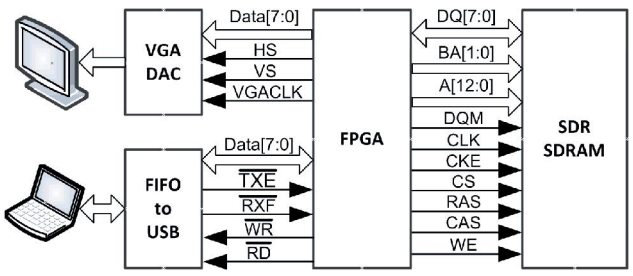


Fig. 3 – Interconnection between the FPGA device and SDRAM, VGA DAC and FIFO-to-USB module.

4. HDL LIBRARY FOR VISION AND PERIPHERAL CONTROL TASKS

The internal FPGA architecture is depicted in Fig. 4. It is comprised of the Image Grabber, the Image Processing core, ping pong SRAM buffers, a Data Manager unit as well as the controllers FIFO-to-USB, VGA, SDRAM and SPI. Each sensor requires an Image Grabber, an Image Processing Core and two RAM ping pong buffers. For simplicity, in Fig. 4 we include only one such structure. In the next paragraphs, we describe all the architectural elements in detail.

4.1 IMAGE GRABBER

The principal prerequisite in the image processing system is the frame grabber. Therefore, the frame grabber core constitutes a main functionality in our HDL component library. It can be transported and used in other systems as well. 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 known as horizontal blanking and between consecutive frames as vertical blanking. Control signals FV (Frame Valid) and LV (Line Valid) 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 is considered to be valid and can be read from the FPGA on the next falling edge of PIXCLK.

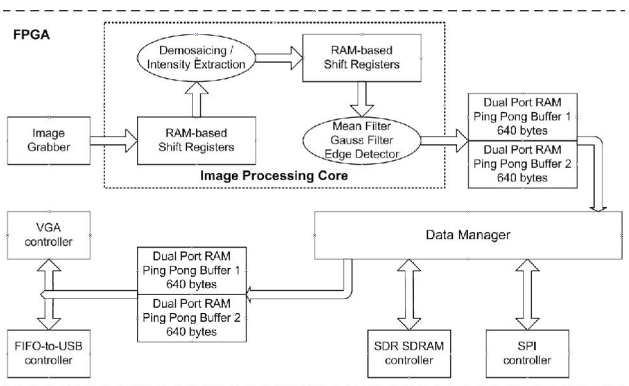


Fig. 4 – FPGA architecture.

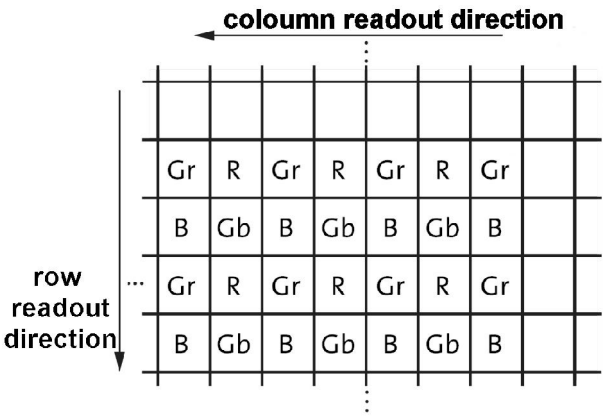


Fig. 5 – Readout order of Bayer encoding.

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 readout order is presented in Fig. 5 and the timing diagram of image readout is depicted in Fig. 6.

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 board. The flow of Finite State Machines (FSMs) is depicted in Fig. 7. In the "Synch" (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', it declares that the sensor is in the vertical blanking interval, which means that the FPGA is now synchronized and can proceed to the "VB" (Vertical Blanking) state. When FV is asserted '1', the FPGA enters into "HB" (Horizontal Blanking) state and if LV is also asserted '1', it enters into the "VID" (Valid Image Data) state.

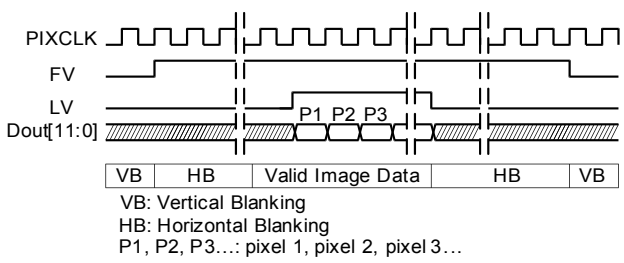


Fig. 6 – Timing diagram of an image readout.

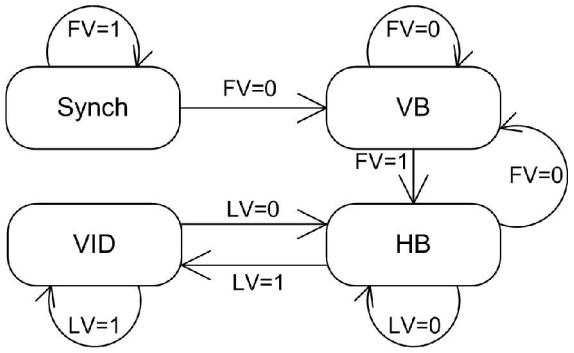


Fig. 7 – Image readout FSM.

Now, the device can read sensor's output, as sensor data is considered valid at every falling edge of PIXCLK. When the sensor completes transmission of the first row's pixel data, it asserts LV to '0' and the FPGA device enters into "HB" state. When the horizontal blanking interval is over, LV is asserted again to '1' and the FPGA enters back to the "VID" state. This sequence will continue until the sensor completes the transmission of the last image row. Afterwards, LV and FV are asserted '0' consecutively and the FPGA enters first into "HB" state and finally into "VB" state. The FPGA device remains there until the sensor starts sending the next frame.

4.2 IMAGE PROCESSING CORE

The Image Processing Core is responsible for the main image processing task. In the present version of our HDL library, a Mean filter, a Gauss filter and an Edge detector have been implemented. These functions are required in many applications as basic pre-processing stages.

Below, the 3x3 kernel matrices applied on input image are quoted. Kernel *M* is for mean filter, *G* is for Gauss filter and *S1*, *S2* are Sobel masks for edge detection.

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

$$S_1=\begin{bmatrix}-1&0&1\\-2&0&2\\-1&0&1\end{bmatrix},\ S_2=\begin{bmatrix}1&2&1\\0&0&0\\-1&-2&-1\end{bmatrix}, \qquad (1)$$

Parallelism requires the pixel intensities of a nxn image area, where the convolution kernel is applied, to be simultaneously available. In our pre-processing stages, we defined n=3. As we have already mentioned, the 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. Parallelism is achieved using RAM-based shift registers. Sensor data is pipelined into shift registers. They are designed in a specific way which outputs image data from 3x3 subwindows in a progressive scan. The operations "Demosaicing/ Intensity Extraction" and "Mean Filter/Gauss Filter/Prewitt Mask", shown as blocks in Fig. 4, are combinational logic functions. The propagation delay of the signals determines maximum processing speed.

The function Demosaicing can be implemented using several ways, as shown in the literature [31]. One simple algorithm is to use the mean value of neighboring colors. Depending on their known color component information, pixels are named as Gr, Gb, R or B, as shown in Fig. 5. Table 1 presents how full color information is calculated in each case. After the Demosaicing procedure completes calculations for a pixel that belongs to row i and column j, grayscale intensities are extracted as in (2):

$$I(j,i) = (Red + Green + Blue) / 3, \qquad (2)$$

Intensity calculations and filtering are carried out concurrently, while the FPGA is reading subsequent pixel data. When the FPGA reads data from row i, it also completes processing on previous rows. From now on we follow the convention of naming a pixel that is at the i^{th} row and the j^{th} column as $p_{j,i}$. Let us consider applying the 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. Let us focus on the 4x4 window that is highlighted with bold borders. 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 perform demosaicing on $p_{j-1,i-1}$ and then extract the grayscale 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 read $p_{j,i}$, it will be able to perform filtering computations on $p_{j-2,i-2}$. Hence, in order the FPGA to be able to apply the processing algorithm on pixel $p_{j-2,i-2}$, it must have available the pixel information in the bold 4x4 window of Fig. 8.

Table 1. Extracting full color information – Demosaicing.

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

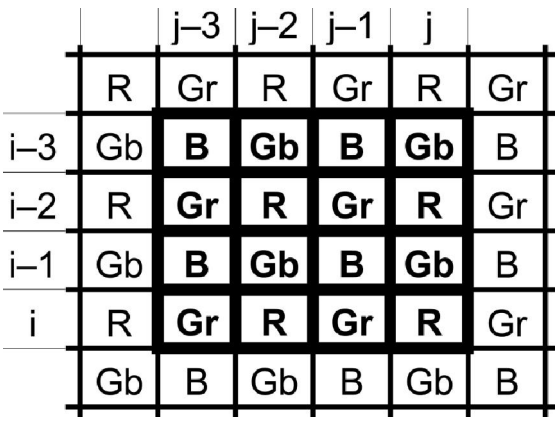


Fig. 8 – Image window necessary for 3x3 convolutions.

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

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

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 (4):

$$I'_{j-2,i-2} = \frac{1}{9}\sum_{x=1}^{3}\sum_{y=1}^{3} m_{x,y} \ . \qquad (4)$$

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

$$G_{p_{j-2,i-2}} = \begin{bmatrix} 1\cdot I_{j-3,i-3} & 2\cdot I_{j-2,i-3} & 1\cdot I_{j-1,i-3} \\ 2\cdot I_{j-3,i-2} & 4\cdot I_{j-2,i-2} & 2\cdot I_{j-1,i-2} \\ 1\cdot I_{j-3,i-1} & 2\cdot I_{j-2,i-1} & 1\cdot I_{j-1,i-1} \end{bmatrix} \qquad (5)$$

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 (6):

$$I'_{j-2,i-2} = \frac{1}{16}\sum_{x=1}^{3}\sum_{y=1}^{3} g_{x,y} \ . \qquad (6)$$

Edge detector implementation is slightly different from Mean and Gauss filters. First, two Sobel matrices S_1 and S_2 are calculated for the pixel $p_{j-2,i-2}$.

$$S_{(1)p_{j-2,i-2}} = \begin{bmatrix} -1\cdot I_{j-3,i-3} & 0\cdot I_{j-2,i-3} & 1\cdot I_{j-1,i-3} \\ -2\cdot I_{j-3,i-2} & 0\cdot I_{j-2,i-2} & 2\cdot I_{j-1,i-2} \\ -1\cdot I_{j-3,i-1} & 0\cdot I_{j-2,i-1} & 1\cdot I_{j-1,i-1} \end{bmatrix} \qquad (7a)$$

$$S_{(2)p_{j-2,i-2}} = \begin{bmatrix} 1\cdot I_{j-3,i-3} & 2\cdot I_{j-2,i-3} & 1\cdot I_{j-1,i-3} \\ 0\cdot I_{j-3,i-2} & 0\cdot I_{j-2,i-2} & 0\cdot I_{j-1,i-2} \\ -1\cdot I_{j-3,i-1} & -2\cdot I_{j-2,i-1} & -1\cdot I_{j-1,i-1} \end{bmatrix} \qquad (7b)$$

Considering every element of matrix S_1 and S_2 as $s_{(1)x,y}$ and $s_{(2)x,y}$ where x=1,2,3 and y=1,2,3 the output image is produced as in (8):

$$I'_{j-2,i-2} = \begin{cases} 0, \left|\sum_{x=1}^{3}\sum_{y=1}^{3} s_{(1)x,y}\right| + \left|\sum_{x=1}^{3}\sum_{y=1}^{3} s_{(2)x,y}\right| < T \\ 255, \left|\sum_{x=1}^{3}\sum_{y=1}^{3} s_{(1)x,y}\right| + \left|\sum_{x=1}^{3}\sum_{y=1}^{3} s_{(2)x,y}\right| > T \end{cases} \qquad (8)$$

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. This method 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.

All the above computations are implemented in VHDL. Most of the divisions were implemented performing right shifts. Instead of division by 3 in (2) and by 9 in (4), we approximate the quotient by multiplying with the divisor's reciprocal. In (2), we multiply the sum of color component information with 341 and then we divide with 1024 performing right shift ten times. In (4), we multiply with 113 and then we proceed with ten right shifts, accordingly.

Using TimeQuest Timing Analyzer tool, it is found that the propagation delay-times (5) limit the maximum processing rate of the Image Processing Core to approximately **270 fps.** No further optimization was attempted, since the maximum supported frame rate of the overall system is actually lower than the above limit, due to the bottleneck introduced by other components.

4.3 DATA MANAGER

The processed pixel data is extracted from Image Processing Core and then it is stored to a dual port RAM, with a capacity of 1280 bytes. This memory is divided into two ping-pong buffers. Each buffer is sufficient for one row of pixel data. While data is being saved to buffer 1, the contents of buffer 2 are being transferred to the Data Manager. When ping pong buffer 1 is full, subsequent pixel data is saved to ping-pong buffer 2 and vice versa. It is important that data must have been transferred before the same memory location is accessed again. RAM ping pong buffers use PIXCLK for write accesses and the on-board 50 MHz clock for read accesses. Also the rest of the logic uses the 50 MHz oscillator as system clock.

The Data Manager consists of various VHDL processes that are responsible for the communication with all other implemented cores inside the FPGA. It receives data from dual port RAM ping pong buffers and transmits them to the SDRAM controller in order to have a full frame stored in the external SDRAM. Moreover it reads image data from the SDRAM controller and transmits to a second dual port RAM. This RAM is also divided into two ping-pong 640 byte buffers. They are used in conjunction with VGA or FIFO-to-USB controllers for the display of video frames on a screen and for the transfer of images to a host computer respectively. The Data Manager also exchanges data with SPI controller for communication with the microcontroller. In general, it is a crossroads for data streaming in the overall system.

4.4 SDR SDRAM CONTROLLER

An optimized SDR SDRAM controller was developed in VHDL for communication between the FPGA chip and external RAM. It is designed as a

fully optimized custom module and therefore a brief description is required. The controller handles all low level operations such as bank/row activation, design aspects of an SDRAM controller can be found in [32]. Our controller provides a user friendly communication interface to interconnect with other cores inside FPGA. The designed module is presented in Fig. 9a. The input signals *wr_req* and *rd_req* are used to send write or read requests to the controller.

Table 2. SDRAM controller truth table.

Input bits		Clock	Status bits			Function
rd_req	wr_req	clk	Rd_valid	sdram_busy	wr_dis	
X	X	X	X	1	X	Busy – write/read requests are ignored
0	0	↑	X	0	X	NOP
1	0	↑	X	0	X	Start read sequence – *addr[24:0]* must have SDRAM access location
X	0	↑	1	0	X	*DQ[7:0]* word has valid data to be read
0	X	↑	X	0	1	Write requests are prohibited
0	1	↑	0	0	0	Start write sequence – signal *dqin[7:0]* must have input data *addr[24:0]* must have SDRAM access location
1	1	↑	X	0	0	Write request is serviced, it has higher priority

X: Do not care.

The *dqin[7:0]* bus is data input and *addr[24:0]* bus is address input. The signals *sdram_busy*, *wr_dis* and *rd_valid* are outputs and indicate the status. The rest of the signals are used to interface with SDRAM. The controller's truth table is presented in table 2.

In Fig. 9b, the block diagram of controller's functionality is shown. After power is on, an initialization procedure follows. It consists of the required auto-refresh cycles, sets the Mode Register and precharges SDRAM to end up in the "Idle" state. When a request from some external function arrives, the controller activates the appropriate bank and row and proceeds with the corresponding write or read action. It stays on that state to allow consecutive operations until the next row/bank has to be activated or a different execution is requested. This will force the controller to precharge the row and/or bank leading to the "Idle" state. Afterwards, the appropriate activation of a bank/row is performed to complete the target operation.

4.5 SPI CONTROLLER

A communication interface between the FPGA and the microcontroller is necessary for data exchange.

SPI is chosen because it is simple, demands few I/O pins and is available on the microcontroller side. Thus an optimized master SPI controller has been developed in VHDL for the FPGA. Design aspects of the SPI controller can be found in [33]. The necessary low level operations such as clock extraction, data registers shift, etc. are realized by the controller. External functions need only to manage appropriately the interface signals. The SPI controller module is depicted in Fig. 10a. Inputs and outputs to the controller are the SPI signals SCLK (Serial Clock generated by the master), MISO (Master Input Slave Output) and MOSI (Master Output Slave Input), as well as the interfacing signals for communication with other functions inside the FPGA.

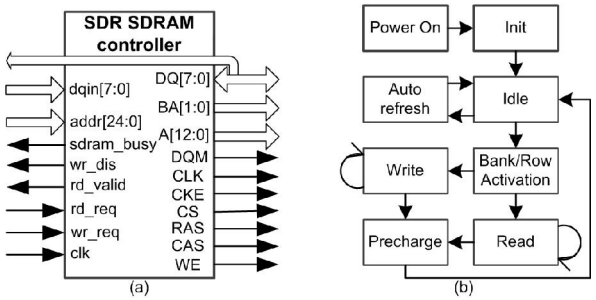


Fig. 9 – (a) The SDRAM controller module. (b) The block diagram of the controller.

The heart of the controller consists of two shift registers. Register *spitxsr* is the transmit and *spirxsr* is the receive shift register. Transmission starts when data is loaded in *spitxsr*. In every clock cycle (SCLK), one bit is shifted out from the transmit shift register and one bit is shifted into the receive shift register. A load to the *spitxsr* is performed when an external function writes data to the *spitx* input register. If *spitxsr* is empty an immediate transfer to that register occurs and communication starts. When a transaction is successfully completed, input data is loaded from *spirxsr* to the *spirx* output register. There is also an internal buffer, named *spitxb*, which can be loaded before previous transmission is

finished. This allows next transaction to start immediately. The SPI clock has been configured at 25 MHz which supports bidirectional communication at 25 Mbits/s. The block diagram of SPI controller is illustrated in Fig. 10b. The truth table is presented in Table 3. Signal *spirx_req* is used when data is read from output buffer by an external function. Signal *spitx_req* is
used to load data to the transmit buffer or shift register. Signals *spirxf*, *spitxbf* are status bits indicating that the output register has valid data to be read and that transmit buffer is full, respectively.

Table 3. SPI controller truth table.

Input bits		Clock	Status bits		Function
spirx_req	spitx_req	clk	spirxf	spitxf	
0	0	↓	X	X	NOP or proceed with current transfer
1	0	↓	1	X	Read data from spirx register
0	X	↓	X	1	Transfer in progress – transmit buffer is full
X	1	↓	X	0	Transmit request, data from spitx are loaded to spitxb

X: Do not care.

4.6 VGA CONTROLLER

Real-time applications require processing of many frames per second. Displaying data on a screen can be a good solution for evaluation purposes or supervision of what the machine "sees". A VGA screen has satisfactory refresh rate and therefore it is suitable to display plain or processed frames. For the current research, a VGA controller that supports resolution of 640x480 pixels at a refresh rate of 60 Hz is developed. The controller is responsible to produce synchronization signals and to send pixel data to the VGA DAC module. More details on the implementation of a VGA controller can be found in [34]. The custom designed module is presented in Fig. 11a. In order to preserve I/O pins from the FPGA, the controller uses one 8-bit bus to send data and consequently it supports only gray-scale images. External functions need to interface the following signals:

vga_in[7:0] – Data to be sent to VGA DAC which correspond to pixel intensity.

vga_req – Output signal indicating that the controller needs data.

clk – Controller clock - a frequency of 25 MHz is required for resolution 640x480 pixel, at 60 Hz.

Every time *vga_req* signal is set, external functions must feed the controller with next valid pixel data. The controller does not include internal buffers in order to be as compact as possible.

In Fig. 11b the controller's FSM diagram is depicted. Every new frame starts with the controller in state A and after 64 us enters in state B. States C, D, E and F are responsible for managing every row. The row sequence is initiated with the controller in C and then in D managing Horizontal Synchronization (HS) signal appropriately. Thereafter, the controller enters in E for 640 clock cycles receiving row pixel data. When the row ends the controller enters in F state for a small delay. If it is not the end of a frame then the sequence of states C, D, E and F is repeated. If it is the end of a frame then the controller enters in G state, where it blanks its output. After the G state, it goes again to A state, for the next frame to be displayed.

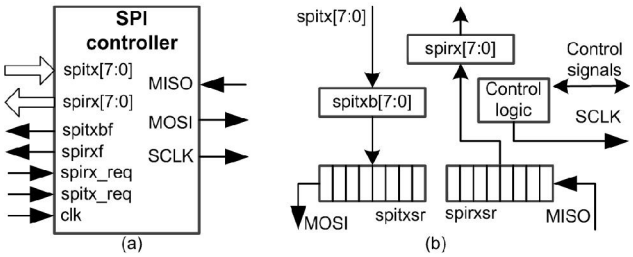


Fig. 10 – (a) The SPI controller module. (b) SPI controller block diagram.

4.7 FIFO-to-USB CONTROLLER

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 can just be stored for further evaluation. By transmitting results to a personal computer, debugging purposes can be served as well. This capability is incorporated in our system providing USB connectivity with a computer. The system includes the UM232H FIFO-to-USB module, which is responsible for all low and high level operations for bidirectional communication with computer's USB port. It uses a First In First Out buffer to transmit or receive 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. Since we mostly send data to the computer, we concentrate on the write interface.

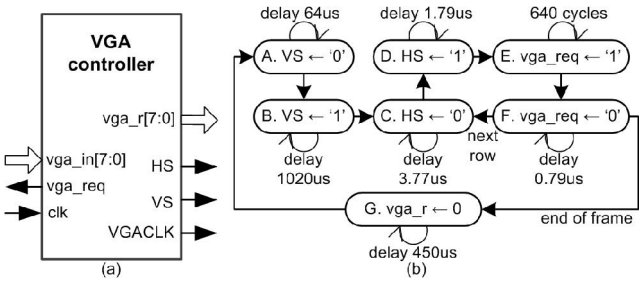


Fig. 11 – (a) The VGA controller module. (b) VGA controller's FSM.

The timing diagram for a typical write sequence is shown in Fig. 12. The FPGA device must assert interface signals according to the write sequence in order to transmit. Apart from the correct order, signal assertion is subject to timing constraints. Timing constraints are defined by the manufacturer and for successful transactions they must be adhered precisely. Signal timing constraints related to the write procedure are quoted in Table 4. A suitable FIFO-to-USB VHDL controller has been implemented, as a Finite State Machine. The state machine is optimized to operate at 50MHz, derived from the external crystal oscillator. The write sequence is depicted in Fig. 13.

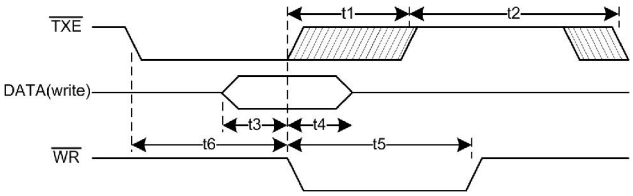


Fig. 12 – FIFO-to-USB write sequence.

Table 4 Timing constraints for FIFO-to-USB write sequence.

Time	Description	Min	Max	Units
t1	WR active to TXE inactive	1	14	ns
t2	TXE active to TXE after WR cycle	49		ns
t3	DATA to WR active setup time	5		ns
t4	DATA hold time after WR inactive	5		ns
t5	WR active pulse width	30		ns
t6	WR active after TXE	0		ns

At the beginning, the FPGA controller stays at the "Idle" state. In this state, no data is sent. CMD constitutes an internal control signal, which is asserted to '1' when the FPGA wants to transmit data. When the CMD signal is asserted '1' from a process, then the controller enters the "Send" state. It stays in that state for as long as the TXE signal is asserted '1'. The TXE is an output signal and when it asserts '1', it indicates that UM232H module is busy or the internal FIFO buffer is full, as a result of a previous transmission. As long as the TXE signal is asserted '1', every write attempt will be ignored. 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. States "Intermediate 1" and "Intermediate 2" provide two clock cycles delay, according to the timing requirements of the write sequence.

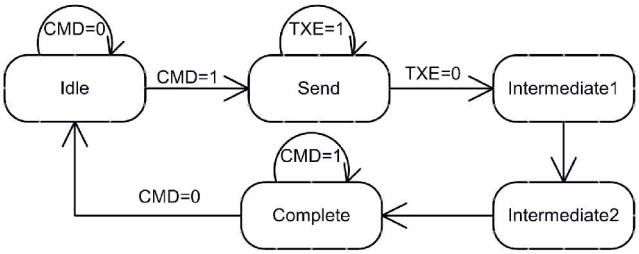


Fig. 13 – FIFO-to-USB write sequence FSM.

Finally the process 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 machine sequence is designed to be fully compatible with UM232H write sequence.

5. MICROCONTROLLER FIRMWARE AND COMPUTER SOFTWARE

The microcontroller constitutes a co-processing unit in the system. The FPGA device exchanges data with the PIC MCU. The microcontroller has reserved two buffers in its internal RAM. Each buffer's capacity is about 60 Kbytes, capable to hold more than 90 rows of image data. In the first buffer, data is saved as it is received from the SPI. After processing, results are stored in the second buffer. Thereafter, the microcontroller sends the contents of the second buffer back to the FPGA. While the microcontroller receives data that belongs to a current row it can simultaneously transmit processed pixels of previous rows. SPI uses DMA to access RAM without CPU intervention. When a transfer is completed, a request is passed to the DMA controller and a transfer from the SPI receive register to a location of the first RAM buffer is performed. At the same time, a second request is passed to the DMA controller and a transfer from a location of the second RAM buffer to the SPI transmit register is accomplished. The above scheme is repeated every time a SPI transaction is

performed. Thus, the microcontroller's CPU is relieved from communication procedures and is devoted entirely to the processing task. At this stage of our research, only the communication interface is implemented exchanging test data between the FPGA and the microcontroller. In the future, when more complex tasks will be requested, the PIC MCU will undertake essential operations.

The microcontroller is also used for interconnection with the CMOS image sensors through I^2C bus. It configures the resolution of the captured image at 640x480 pixels, the horizontal and vertical blanking intervals as well as the shutter width. It also manages PIXCLK which defines sensor's output pixel data rate. Taking into consideration the bottleneck introduced by SDRAM accesses, PIXCLK was defined at 24 MHz. Blanking intervals and shutter width were configured appropriately and as a result the sensor produces images at 30 fps which is the maximum supported frame rate for the overall system.

On the host computer side, a software application has been developed in Visual Basic. The application receives image data and displays them on the screen. It is also capable of saving image frames to disk for future research or for evaluation purposes. It uses D2XX vendor's driver for USB communication. At present, the supported transfer speed to the computer is 4 fps, however, the proposed system is capable of higher transfer rates. As a future project, a more sophisticated computer interface will be developed to support real-time transfer rates.

6. SYSTEM EVALUATION

The system uses two clock sources. The first clock source is PIXCLK which is derived from the CMOS sensor and is the main clock for the Image Processing Core. The second clock source is the on board oscillator with frequency 50MHz. This clock is used for every other synchronous operation in the system. The maximum frequency of PIXCLK that meets the timing requirements of the implemented image processing tasks was found to be 83 MHz. This value was calculated using Altera's timing analysis tool for the Mean Filter, Gauss Filter and Sobel Edge Detector. This means that the Image Processing Core is capable of processing 83 Mpixel/sec or 270 fps for a resolution of 640x480 pixels of gray-scale images. This frame rate could be further increased by optimizing the circuit realized by the synthesizer. However, this would be a meaningless effort, since the overall performance is reduced by other elements. The Data Manager, the FIFO buffers and the internal peripheral controllers are clocked by the 50 MHz oscillator. Since these circuits execute only data transfers without applying any processing, they can support higher clock frequencies. A higher frequency would increase performance but it would also increase power consumption and radiation emitted from the board leading to EMC or EMI issues. Thus a compromise in performance is considered. The main bottleneck is found in SDRAM operation. After pixels are read and processed, pixel data is stored in SDRAM. When output FIFO buffers are empty, pixel data is read from SDRAM and buffers are loaded until they are full. External SDRAM is also clocked at 50 MHz with a signal derived from the FPGA. This means that it is capable to read or write or combine reads/writes at a maximum rate of 50 Mpixel/sec. In practice, the speed is lower since internal SDRAM operations such as auto-refresh cycles, bank/row activation, precharge operation and etc. consume considerable number of cycles. When a fast interface is realized for image display, then maximum write speed is reduced. When a slower output interface is applied then write speed can be increased. When the VGA interface is used, a bandwidth of 25 Mpixel/sec is required. Our experimental results show that the image sensor's maximum clock frequency, at which the system operates normally, is 24 MHz. Considering the horizontal and vertical blanking intervals, our system is capable of displaying plain or processed video data at 30 fps which stands for real time performance. When our system is connected with a host computer then it is capable of processing at higher speeds but it can display plain or processed video data at the rate allowed by the USB interface.

As we have already mentioned, we did not proceed to further optimizations for the maximum theoretical speed of the Image Processing Core. We have only replaced divisions with multiplications with divisor's reciprocals and let the development tools to synthesize and fit the circuit inside the FPGA, using only logic elements. Hardware multipliers have been left unused for future tasks and necessary multiplications are performed using only LEs. The total resources needed to implement our system are presented in Table 5. These resources include two frame grabbers, two Image Processing Cores, two RAM buffers for storing plain or processed data from both sensors, the Data Manager, two RAM buffers for use with output device and the SDR SDRAM/SPI/VGA controllers. When the VGA controller is replaced with FIFO-to-USB controller, in order to send frames to the computer, two more I/O pins are required, but fewer logic elements are allocated.

In Table 6, the power dissipation of the system in operation is presented. Low power consumption is important, since our vision system is designed for robotic applications that often run on batteries. The FPGA consumes about 190 mW. The

Microcontroller and the SDRAM demand about 430 mW, which is a considerable consumption. This is a trade off for having a co-processing unit incorporated in the system. VGA DAC consumption is 231mW. However, this is a supplementary function used for evaluation and demonstration purposes and will be excluded in a final application. The main drawback is the power consumption of the CMOS header board. It is supplied with additional circuitry which demands a total of 500mW. For a stereo vision system, the power consumption of both sensors is 1 W. The choice of the specific sensor header board aims to support high performance in future research. However, if power consumption and cost are essential elements in an application, a replacement would rather be considered. The Toshiba TCM8230MD color CMOS image sensor may constitute a candidate, since it is inexpensive and consumes only 60mW.

A main advantage of the system is the ultra-low cost with only a small degradation in performance. Not only the processing module, but the overall system achieves real time speed and it provides comparable results with other systems that cost many thousands of euros. Excluding the CMOS image sensor the system costs less than 80 euros. By using two TCM8230MD devices instead of the 5 Mpixel sensors, the total cost of the complete stereo vision system does not exceed 100 euros. To the best of our knowledge, no other FPGA-based system presented in the literature, that is intended to be used for image processing, maintains the cost at such a low level. Since the system is designed as a project open to further development, it is fully implemented in standard VHDL, free of dependencies on Intellectual Property cores, with all its parts optimized for minimum resource usage.

The proposed system is intended to be used in advanced robotic vision projects. The HDL library, introduced in this article, will be expanded to include necessary processing tasks, such as stereo vision, feature extraction, optical flow calculation etc. On the hardware part, some enhancements are also considered. Since the main bottleneck of the system appears to be in the communication with the SDR SDRAM, a replacement of this memory with a DDR is considered. In this way, the overall speed can be doubled using the same main clock frequency and a processing rate of 60 fps can be achieved.

Another future enhancement is the implementation of a high speed wireless link for transferring image data to a base station. VGA and USB connectivity will be removed from the mobile unit and will be incorporated into the base station.

7. EXPERIMENTAL RESULTS

In this section we present image results monitored while the overall system is in action. All images have resolution 640x480. The image in Fig. 14 (a) is in Bayer encoding. It is transmitted to the host computer as captured by the image sensor, without any further processing. The image in Fig. 14 (b) is derived after demosaicing the first image. Fig. 15 shows results from the Edge Detector, as received by the host application. The image in Fig. 15 (a) is the output of the Frame Grabber in gray-scale. Fig 15(b) is the image produced applying the Sobel masks and the thresholding procedure. The display of the transmitted image on a VGA screen is captured by a camera and is demonstrated in Fig. 16. In the first picture the plain image is shown, while in the second the corresponding edge detector results are presented. The test rig of the developed system is illustrated in Fig. 17.

Table 5. Resource usage.

Resources	Available	Used	Percentage
Total LE	22320	2423	10,86%
Total pins	80	69	86,25%
Total memory bits	608256	51360	8,44%
Embedded 9-bit multipliers	132	0	0%
Total PLLs	4	0	0%

Table 6 Power dissipation.

Structure	Current drawn	Power consumption
FPGA core	17mA at 1.2V	20,4 mW
FPGA analog circuitry supply	28mA at 2.5V	70 mW
FPGA I/O	30mA at 3.3V	99 mW
Microcontroller	100mA at 3.3V	330 mW
SDRAM	30mA at 3.3V	99 mW
VGA DAC	70mA at 3.3V	231 mW
CMOS sensor header boards	2x100mA at 5V	1000 mW
Total power dissipation for a stereo vision system in operation		≈1850mW

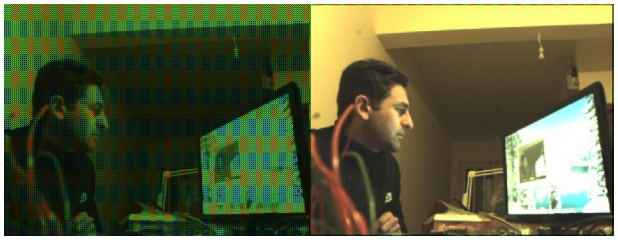


Fig. 14 – (a) Image in Bayer encoding. (b) Image after demosaicing.

Fig. 15 – (a) Gray-scale image. (b) Edges of gray-scale image.

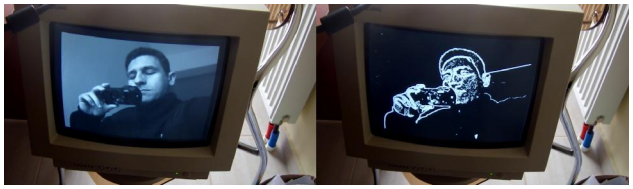


Fig. 16 – (a) Gray-scale image on VGA screen. (b) Edges of gray-scale image on VGA screen.

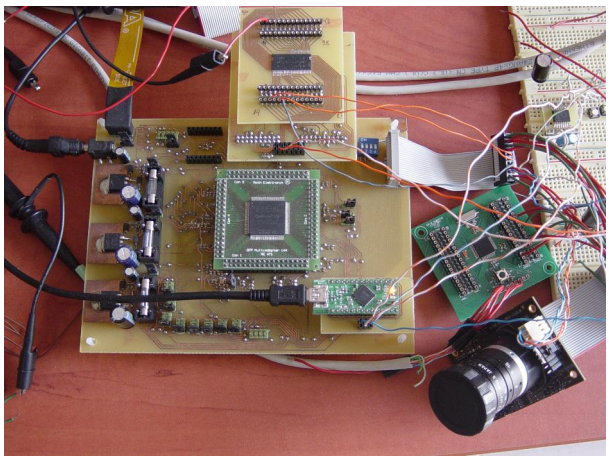


Fig. 17 – Test rig of the proposed system.

8. CONCLUSION

In this paper, the design and implementation of a high performance video-processing FPGA/ microcontroller board is presented. The system is based on a flexible modular architecture comprised of a Cyclone IV Altera FPGA device and a 32-bit PIC microcontroller. FPGA resources are allocated to implement video processing functions and basic peripheral controllers, while the MCU is able to support peripheral functions and/or co-processing tasks. The system includes a frame grabber suitable to interface with two CMOS image sensors. It also includes external SDRAM, able to store captured frames, a VGA DAC module for fast image display on a screen and a FIFO-to-USB module, providing connectivity to a host computer for further processing. The Image Processing Core processes up to 270 fps and the overall system in action has a proved performance of 30 fps. The controllers along with basic video processing tasks require only a small fraction of the available resources. The cost of the proposed board is low compared to existing commercial video kits or other implementations published in the literature. The system is expandable and is intended to host demanding machine vision applications. The schematic and PCB designs and the HDL source code of designed tasks are open to the vision and robotics community for academic purposes and will be further developed. Full access to the necessary files for the reproduction of the system is granted by addressing a request by e-mail to the corresponding author.

6. REFERENCES

[1] H. Hagiwara, K. Asami, and M. Komori, FPGA implementation of image processing for real-time robot vision system, in *Proceedings of the 5th International Conference on Convergence and Hybrid Information Technology, ICHIT'2011*, Daejeon, Korea, (2011), pp. 134-141.

[2] S. Venugopal, C. R. Castro-Pareja, and O. Dandekar, An FPGA-based 3D image processor with median and convolution filters for real-time applications, in *Proceedings of the SPIE-IS and T Electronic Imaging - Real-Time Imaging IX*, San Jose, CA, (2005), pp. 174-182.

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

[4] OpenCV (Open Source Computer Vision), 2013, available online on http://opencv.org/.

[5] Nvidia, CUDA: Parallel computing platform, 2013, available online on http://www.nvidia.com/object/cuda_home_new.html, accessed June 2013.

[6] J. A. Kalomiros and J. Lygouras, Design and evaluation of a hardware/software FPGA-based system for fast image processing, *Microprocessors and Microsystems*, (32) 2 (2008), pp. 95–106 [doi: 10.1016/ j.micpro.2007.09.001].

[7] D.-T. Lin, M.-C. Lin, and K.-Y. Huang, Real-time automatic recognition of omnidirectional multiple barcodes and DSP implementation, Machine Vision and Applications, (22) 2 (2011), pp. 409-419.

[8] C. González, S. Sánchez, A. Paz, J. Resano, D. Mozos, and A. Plaza, Use of FPGA or GPU-based architectures for remotely sensed hyperspectral image processing, *Integration, the VLSI Journal*, (46) 2 (2013), pp. 89-103.

[9] J. Fowers, G. Brown, J. Wernsing, and G. Stitt, A performance and energy comparison of convolution on GPUs, FPGAs, and multicore processors, *Transactions on Architecture and Code Optimization*, (9) 4 (2013), art. no. 25.

[10] H. Hou, W. Zhang, D. Huang, and T. Zhang, Design and realization of real-time image acquisition and display system based on FPGA, in *Proceedings of the International Conference on Mechanical Engineering and Technology, ICMET'2011*, 125, London (2012), pp. 565-573.

[11] C. Li, Y.-L. Zhang, and Z.-N. Zheng, FPGA-based CMOS image acquisition system, *Communications in Computer and Information Science*, (34) (2009), pp. 122-127.

[12] B. Yan, Y. Sun, F. Ding, and H. Yuan, Design of CMOS image acquisition system based on FPGA, in *Proceedings of the 6th IEEE Conference on Industrial Electronics and Applications, ICIEA'2011*, Beijing (2011), pp. 1726-1730.

[13] D. Huang, T. Zhang, H. Hou, and W. Zhang, Design of system for high-frame frequency image acquisition and remote transmission, in *Proceedings of the 3rd International Asia Conference on Informatics in Control, Automation and Robotics, CAR'2011*, Shenzhen (2011), pp. 743–750.

[14] R. Wang, Z. Mi, H. Yu, and W. Yuan, The design of image processing system based on SOPC and OV7670, *in Proceedings of the International Conference on Advances in Engineering, ICAE'2011*, Nanjing (2011), pp. 237-241.

[15] P. Chalimbaud and F. Berry, Design of an imaging system based on FPGA technology and CMOS imager, in *Proceedings of the IEEE International Conference on Field-Programmable Technology, FPT'04*, Brisbane (2004), pp. 407-411.

[16] P. Premaratne, S. Ajaz, R. Monaragala, N. Bandara, and M. Premaratne, Design and implementation of edge detection algorithm in dsPIC embedded processor, in *Proceedings of the 5th International Conference on Information and Automation for Sustainability, ICIAfS'2010*, Colombo (2010), pp. 8–13.

[17] Y. Said, T. Saidani, F. Smach, M. Atri, and H. Snoussi, Embedded real-time video processing system on FPGA, in *Proceedings of the 5th International Conference on Image and Signal Processing, ICISP 2012*, Agadir (2012), pp. 85-92.

[18] S. Halder, D. Bhattacharjee, M. Nasipuri, and D. K. Basu, A fast FPGA based architecture for Sobel edge detection, in *Proceedings of the 16th International Symposium on VLSI Design and Test, VDAT'2012*, Shibpur (2012), pp. 300-306.

[19] S. Singh, A. K. Saini, and R. Saini, Real-time FPGA based implementation of color image edge detection, *International Journal of Image, Graphics and Signal Processing*, (4) 12 (2012), pp. 19-25.

[20] S. Veni, K. A. Narayanankutty, and M. Raffi, Hardware implementation of edge detection on hexagonal sampled image grids, *International Journal of Computer Applications*, (24) 2 (2011), pp. 29-38.

[21] L. Tian, X. Liu, J. Li, and X. Guo, Image preprocessing of CMOS image acquisition system based on FPGA, International Journal of Digital Content Technology and its Applications, 6 (20) (2012), pp. 130–139.

[22] B. Putz, M. Bartyś, A. Antoniewicz, J. Klimaszewski, M. Kondej, and M. Wielgus, Real-time image fusion monitoring system: Problems and solutions, in *Proceedings of the 4th International Conference on Image Processing and Communications, IPC'2012*, Bydgoszcz (2012), pp. 143-152.

[23] X. Zhang and Z. Chen, SAD-based stereo vision machine on a system-on-programmable-chip (SoPC), *Sensors*, (Switzerland), (13) 3 (2013), pp. 3014-3027.

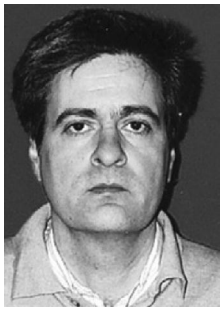
[24] C. Ttofis, S. Hadjitheophanous, A. S. Georghiades, and T. Theocharides, Edge-directed hardware architecture for real-time disparity map computation, *IEEE Transactions on Computers*, (62) 4 (2013), pp. 690-704.

[25] P. Zicari, S. Perri, P. Corsonello, and G. Cocorullo, Low-cost FPGA stereo vision system for real time disparity maps calculation, *Microprocessors and Microsystems*, (36) 4 (2012), pp. 281-288.

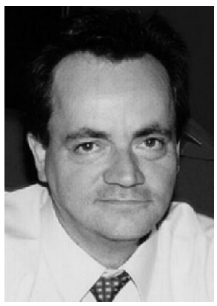
[26] S. Jin, J. Cho, X. D. Pham, K. M. Lee, S.-K. Park, M. Kim, and J. W. Jeon, FPGA design and implementation of a real-time stereo vision system, *IEEE Transactions on Circuits and Systems for Video Technology*, (20) 1 (2010), pp. 15-26.

[27] G. K. Gultekin and A. Saranli, An FPGA based high performance optical flow hardware design for computer vision applications, *Microprocessors and Microsystems*, (37) 3 (2013), pp. 270-286.
[28] S. Zhong, J. Wang, L. Yan, L. Kang, and Z. Cao, A real-time embedded architecture for SIFT, *Journal of Systems Architecture*, (59) 1 (2013), pp. 16–29.
[29] L. Siéler, C. Tanougast, and A. Bouridane, A scalable and embedded FPGA architecture for efficient computation of grey level co-occurrence matrices and Haralick textures features, *Microprocessors and Microsystems*, (34) 1 (2010), pp. 14-24.
[30] B. Tippetts, D. J. Lee, K. Lillywhite, and J. Archibald, Review of stereo vision algorithms and their suitability for resource-limited systems, *Journal of Real-Time Image Processing*, Available on line at http://link.springer.com/content/pdf/10.1007%2Fs11554-012-0313-2.pdf on 1/18/2013.
[31] D. Menon and G. Calvagno, Color image demosaicking: An overview, *Signal Processing: Image Communication*, (26) 8-9 (2011), pp. 518-533.
[32] G. Li and Z. Wu, Design and realization of SDRAM controller based on FPGA, in *Proceedings of the International Conference on Measurement, Instrumentation and Automation, ICMIA'2012*, Guangzhou (2012), pp. 2233-2237.
[33] X. Tian, J. Li, Y. Fan, X. Yu, and J. Liu, Design and implementation of SPI communication based-on FPGA, in *Proceedings of the International Conference on Advanced Engineering Materials and Technology, AEMT'2011*, Sanya (2011), pp. 2658-2661.
[34] F. Ying and X. Feng, Design and implementation of VGA controller using FPGA, *International Journal of Advancements in Computing Technology*, (4) 17 (2012), pp. 458-465.

John V. Vourvoulakis *received his Diploma in 2002 and his MSc Degree in 2004 from the department of Electrical and Computer Engineering of the Democritus University of Thrace. His research interests are in the field of embedded systems based on FPGAs and microcontrollers. He is also working as adjunct teaching staff in the Technological Educational Institute of Lamia, Greece.*



John A. Kalomiros *received the degree of Physics and a MS degree in Electronics from the Aristotle University of Thessaloniki, Greece. His PhD thesis is on the development of vision systems for robotic applications. His research interests include design of embedded systems, machine vision and semiconductors. He is faculty member in the department of Informatics Engineering, Technological Educational Institute of Central Macedonia, Greece.*



John N. Lygouras *received the Diploma degree and the Ph.D. in Electrical Engineering from the Democritus University of Thrace, Greece in 1982 and 1990, respectively, both with honors. From 2012 he is a Professor at the Department of Electrical & Computer Engineering in DUTh.*

His research interests are in the field of robotic systems trajectory planning and execution. His interests also include analog and digital electronic systems and controller design for underwater remotely operated vehicles.