I.J. Modern Education and Computer Science, **2016, 3, 11-21**
Published Online March 2016 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2016.03.02

A Low Cost High Speed FPGA-Based Image Processing Framework

Mohammad Reza Mahmoodi
Department of Electrical and Computer Engineering, Isfahan University of Technology, 8415683111 Isfahan, Iran
E-mail: mr.mahmoodi@ec.iut.ac.ir

Sayed Masoud Sayedi
Department of Electrical and Computer Engineering, Isfahan University of Technology, 8415683111 Isfahan, Iran
E-mail: m_sayedi@cc.iut.ac.ir

Abstract—In this paper, a high-speed and low-cost image processing framework based on MATLAB-FPGA interface is proposed that can be used in researches aiming at developing wide variety of not only image processing tasks but also many signal processing applications. In addition, this new framework could be exploited for several other tasks such as on-chip verification, using PC as an enormous external RAM for FPGA while preserving high speed data access, developing hardware-software co-designs, etc. The communication between FPGA and MATLAB is via 1Gbs Ethernet based on UDP/IP protocol which is very promising for high speed data transmission in point-to-point communications. UDP stack is efficiently designed in FPGA based on a fully pipelined architecture with minimum level of logic in order to reach high performance.. Dynamic data transmission between the UDP stack, memory and an arbitrary image processing module makes it possible to practically simulate, debug and implement most relevant applications. The hardware system is relatively low-cost and it consumes a negligible area of a Spartan-6 LXT45 Xilinx FPGA. Operating at 1 Gb/s, theoretically, the system is capable of processing 132 frames of 640*480 color images in a second. The effectiveness of the system is evaluated by means of both place and route simulation and practical implementation of a skin detection algorithm and a motion detector.

Index Terms—FPGA, MATLAB, Hardware Implementation, UDP Stack, Digital Signal Processing, Image Processing.

I. Introduction

Digital signal processing has remodeled traditional analog signal processing systems as a mature technology. Though analog chip designs used to be implemented on smaller die sizes, but currently, with the noise associated with modern sub-micrometer circuits, digital systems are often much more densely integrated than analog designs and this has been yielded to a compact, low-power, and low-cost designs [1]. DSP (digital signal processing) as a subfield of signal processing is used in numerous applications such as most of associated threads in disparate fields of image processing, audio and speech signal processing, digital communications, biomedicine, etc. Developing both programmable DSP chips and dedicated system-on-chip (SoC) solutions has been an active area of development and research over the past three decades [2]. Three important factors which have undoubtedly affected digital signal processing are development of an efficient way of computing DFT in 60s [3], commercial production of programmable digital signal processors in which calculation of fixed point arithmetic operations in one clock cycle became possible [1], and, finally introduction of modern FPGAs which provides low cost and fast DSP arithmetic implementation using a semi-ASIC architecture.

In order to deal with different design requirements on digital signal processing including image processing system; e.g., video codec as a paradigm application utilized in many real-time applications for which many hardware and software implementations are already presented in literature [4-7]. Although software implementations are easy to realize on general-purpose microprocessors, multiprocessors, microcontrollers, or digital signal processors, their sequentially executing structure is not well suited for fast processing of computational applications such as high resolution compression/video scaling of motion pictures, satellite communication modulator/demodulator, etc [8]. Even though the implementations are various and directly related to the application requirements, in general, they can be grouped in four different categories and also compared as shown in Table. I. In the table, the features are evaluated comparatively. For example, considering the density as a parameter; the 28 nm Virtex-7 Xilinx FPGAs (Field Programmable Gate Arrays) are dense enough to encompass many DSP based designs; however, compared to Application Specific Integrated Circuits (ASICs) (in the same technology), mainly due to the high volume of configuration circuitry in FPGAs the effective density will be much lower (approximately 1 fifth). This difference is denoted in the table by using words Moderate and High, respectively for FPGA and ASIC platforms. It can be deduced from the table that for many applications, FPGAs, compared with other platforms, are remarkable solution

for implementation of DSP algorithms. A brief discussion on this is in the following:

ASICs seem to be very promising solution in terms of both performance and power [9]. However, this type of implementation is not a straightforward solution mainly because of high cost of prototyping and long time to market. FPGAs on the other hand have some features which make them very favourable. Their intrinsic parallel structure, relatively low cost, relatively lower time to market, and finally reconfigurable architecture make them remarkable solution for many communicational and processing algorithms [10]. In addition, their much higher flexibility and lower turnaround time is very attractive when comparing these devices with ASICs [11].

Table 1. Different Implementation Platform's Features

Platform	ASIC	DSP	General Purpose Processors	FPGA
Performance	**Highest**	**Moderate**	**Lowest**	**High**
Unit Cost (Prototyping)	**Highest**	**Moderate**	**Lowest**	**Moderate**
Power	**Lowest**	**Moderate**	**Highest**	**Moderate**
Flexibility	**Lowest**	**Moderate**	**Highest**	**High**
Design Effort (Complexity)	**Highest**	**Moderate**	**Lowest**	**Moderate**
Time-to-Market	**Highest**	**Moderate**	**Low**	**Moderate**
Density	**High**	**Low**	**Highest**	**Moderate**

Whenever a system with moderate cost and performance is needed, specialized microprocessors can be utilized. DSP chips are capable of both fixed-point and floating-point arithmetic operations. However, their moderate performance is not suitable for implementation of all signal processing applications. FPGAs have a technical advantage over today's DSPs for example in their silicon technology which can be much more advanced [12]. For instance, modern FPGAs use hardcore dedicated DSP blocks with comparable performance with ASICs. A standard DSP processor (Dual core DSP) running at 300 MHz is capable of performing 2 operations in one clock cycle leading to (300*2)/1=600 MMACS[1]. However, using DSP48A1 dedicated blocks in a Spartan-6 LX45 device running at 250 MHz, leads to 58*250/1=14,500 MMACS due to their both fully parallel and dedicated (ASIC-like) architecture [13]. Of course, whether or not this technical advantage is enough to make FPGAs more attractive than DSPs is heavily depend on application, budget and technology focus.

General purpose computers are the most available, cheapest and simplest choice of design specifically when real-time processing is not in the first priority. Though they seem to have lowest performance and highest power consumption, they are the most common platform for prototyping DSP algorithms mainly due to their simple design flow and low cost of developing. Different software developing platforms are available in computers among which MATLAB (Matrix Laboratory) is a very important and commonly used one. MATLAB is a fourth-generation programming language and it has been a critical tool in developing variety of algorithms. It would be very useful, and sometimes critical, to have a high speed link between general purpose computers and FPGAs as it has many applications though the main focus of this paper is development of an image processing framework.

Developing a MATLAB-FPGA communication link is useful not only in algorithm development, but also in applications which are mostly affiliated with fully or partially development of FPGA based designs. This platform makes it possible to compare two different implementation of one algorithm in terms of speed, accuracy, etc. Also it is beneficial in tasks such as performing on-chip verification, loading parameters or coefficients of an algorithm to the FPGA (repeatedly), measuring an algorithm's performance or accuracy based on variety of inputs and outputs, etc. For example, in biomedical digital signal processing on FPGA (brain-computer interface or voice processing), various DSP algorithms are implemented on FPGA [14-16] which can leverage the proposed setup as a high speed controlling interface between the master PC and slave FPGA.

Another application is when it is necessary to transfer a huge amount of information and it cannot be stored in FPGA or even off-chip SDRAM memories. For example, in an intensive searching application [17], FPGAs are employed in a highly computational comparison tasks between a sample of data (query) and a large size database. Another application is related to the capability of MATLAB in modeling different digital and analog systems. For example, in linearization of RF power amplifiers, one big branch in linearization techniques is based on digital pre-distorters and in general on digital techniques. These digital systems are sometimes implemented using FPGAs [18, 19]. Due to the MATLAB's capability in modeling RF amplifiers as well as other RF systems, using a high speed FPGA-MATLAB interface is a typical solution to test these systems.

A high speed FPGA-MATLAB connection makes it possible to transfer high computational parts of algorithms to the FPGAs, as these parallel architecture devices often outperform software based algorithms. In other words, the system provides a point-to-point FPGA-PC communication which can be directly utilized in implementation of hardware-software co-designs. For example, in a high computational face recognition application [20] it is necessary that the processor access the computer RAM many times while lots of arithmetic operations are executed. But, these algorithms can be partially and effectively implemented in hardware. Another example is implementation of BLAST (Basic Local Alignment Search Tool) algorithm [21].

The aim of this paper is to present a new high speed solution to MATLAB-FPGA interface with negligible hardware resource consumption to provide enough room for execution of other main DSP (particularly image processing) algorithms. The rest of the paper is as follows.

[1] MMACS = Millions of Multiply-accumulate per second (measure of DSP performance)

The design of the system is elucidated in section II and two applications of the proposed system i.e. implementation of skin detection and motion detection algorithms and their practical results are provided in section III. Finally, conclusion is provided in section IV.

II. System Level Design

The MATLAB-FPGA interface is a couple node system consisting of a PC and a FPGA. In Fig. 1, the overall architecture of the system is depicted focusing on image processing applications. By revising in minor parts of the FPGA design and input/output external devices, implementation of other relative applications is possible. MATLAB initiates the entire operation. The Software core which is developed in PC is capable of capturing the image from either an external device camera or computer hard disk and sending it via the communication link in a user-friendly manner. These packets are sent over a high speed one gigabit Ethernet technology. In the FPGA, input packets are received and depend on the content of the data and packets, further processes will be executed. Two standard protocols are utilized here; the first one is ARP (address resolution protocol) and the other one is the UDP which carries the main data. The former one is prerequisite in order to establish the connection; in fact, after global reset of the FPGA applied and execution of first MATLAB instructions, PC broadcasts ARP packets. ARP protocol is employed in order to convert the IP address to a physical address. FPGA should be able to transmit ARP request and reply frames to inform the PC of its own address. The latter protocol i.e. UDP, due to its desirable features, is used for transferring the main signal.

The main operation starts by loading data into the FPGA and it will be stored in an off-chip high dense SDRAM. Before buffering the data, it is possible to perform particular kind of operations. For example, it is possible to implement an in-pixel processor to perform specific task (e.g. filtering, edge detection, etc). Most DSP applications require large memories for storage and buffering data and using FPGA internal RAMs is not efficient at all. SDRAMs offer a high capacity (up to a number of gigabits) as well as high levels of bandwidth for data transmission with the cost of area, power and management complexity. The latter one is critical since different modules should be able to access SDRAM at the same time, and the former is essential as in many applications such as image processing algorithms, SDRAM should be able to store a huge amount of data. SDRAM's operation is controlled by means of a dedicated memory controller inside the FPGA. Any module which is required to access SDRAM's data is associated with the memory controller. The main module which is the hardware implementation of the DSP algorithm has also a dedicated access to the memory controller; thus, the data will be written and read back whenever it is needed. A controller is provided to control the overall operation and manage memory addressing tasks. An optional interface to output device (here it is a monitor) is also utilized to observe the results if desired, even though it is possible to send the processed data back to the software core.

A. Ethernet Connection

In order to provide the PC-FPGA connection, Ethernet technology is utilized. The USB 2.0 can provide approximately 500 Mb/s transfer rate, but it is not enough for many applications and also it is not applicable for long length cables. Higher versions such as USB 3.0 and USB 3.1 provide much higher bandwidth, but they are not always compatible with current systems. RapidIO, PCIe and Ethernet are other standard choices. Among them, Ethernet (defined and standardized in IEEE 802.3-2008) seems more preferable. It is fast enough for many applications, has low cost, and is available almost wherever there is a computer. PCIe is the fastest interconnect technology available for FPGA-PC connection, but it has several drawbacks [22]. In past, for a simple point to point communication, the need for additional circuits could increase the overall cost of the system and a processor was required to implement a network stack [23]. Currently, FPGAs make it possible to cost-effectively implement the stack very straightforward.

Besides choosing a hardware platform, e.g., FPGA, and interconnect technology, e.g., gigabit Ethernet, a protocol is also chosen to run in upper layers (network and transport). Regarding protocols and standards, several points should be considered before designing the stack. In some usages, the accuracy has strictly the first priority, while in many others it is not that important. In data streaming applications such as real-time image processing systems, most of the time, it is crucial that input interface be able to provide several gigabits of bandwidth, but it would be okay if some bytes of data get lost. UDP, due to use of a least protocol mechanism without any special handshaking between the server and clients, seems to be an unreliable protocol. TCP, on the other hand, is a connection-oriented protocol that provides a safe and error free system by sending and receiving acknowledgements and by retransmissions and timeouts [24]. Even though UDP is not suitable for some applications, there are certain situations in which UDP is preferred. According to [25], UDP is very suitable for simple query-response protocols such as DNS and it is stateless, thus, suitable for media streaming applications such as IPTV and video-conferencing. And also, lack of retransmission delay makes this network protocol very useful for real-time applications. An important aspect of a UDP structure compared with a TCP implementation is its less used resources. In fact, due to the more complexity of the TCP, more resources and area is needed for its implementation. Also, the long header of TCP reduces the protocol efficiency compared to that of the shorter UDP header.

The overall architecture of the proposed stack is depicted in Fig. 2. It consists of several sub-modules. The PHY implementation is possible either using on-chip cores such as SGMII LogiCORE or Ethernet 1000Base-X PCA/PMA (offered by Xilinx or other similar cores by other vendors) or employing off-chip ICs such as Marvell Alaska PHY device. The physical layer is implemented

off-chip using BASE-T standard. Marvell Alaska PHY device is used as an interface for Ethernet communications at 10, 100 or 1000 Mb/s speeds. PHY connection to Ethernet cable is through a connector with built-in magnetic. PHY connection to upper layers could be managed through predefined standards such as GMII, RGMII, SGMII, etc which depends on the connection features. GMII interface is defined by IEEE802.3 specification and it is utilized in this stack. When using GMII, the FPGA designer should consider a logic circuit built in FPGA IO blocks to meet timing requirement. Fig. 3 shows the circuit which is used to here both in receiver and transmission interfaces in IOBs. At the GMII transmitter interface, while operating at 1 Gbs, it is necessary that user provides 125 MHz clock (placed on global clock routing) and feed it to the core, client logic and gmii_tx_clk output ports. Of course, this latter signal is first inverted using an ODDR primitive to maximize setup and hold times. Other output signals are registered in IOBs using Double-Data-Rate registers so that the clock, data and other control signals arrive at the same time. In Fig. 4, the timing diagram of the main signals (for an ARP packet including FPGA IP Address) related to the Ethernet transmitter interface is depicted. GMII receiver logical implementation for Spartan-6 includes several IODELAY2 blocks and a BUFIO2 to produce lowest form of clock routing delay. The first output of the BUFIO2 clock (IOCLK) is routed to the IO clock network (IODDR2, IODELAY2, IOSERDES2) and the second one (DIVCLK) drives BUFGs or Clock manager blocks. IODELAY2 blocks should be practically adjusted to fine-tune the setup time and hold time of IOB flip-flops. Delays depend on the routings of the design. Another practical constraint is related to the clock regions in which RX signals are placed; they should be all placed on the same clock region as the RX clock signal. In addition, the output of the DIVCLK of the BUFIO2 drives a BUFG; hence, this clock will be utilized in entire of the receiver part of the UDP stack.

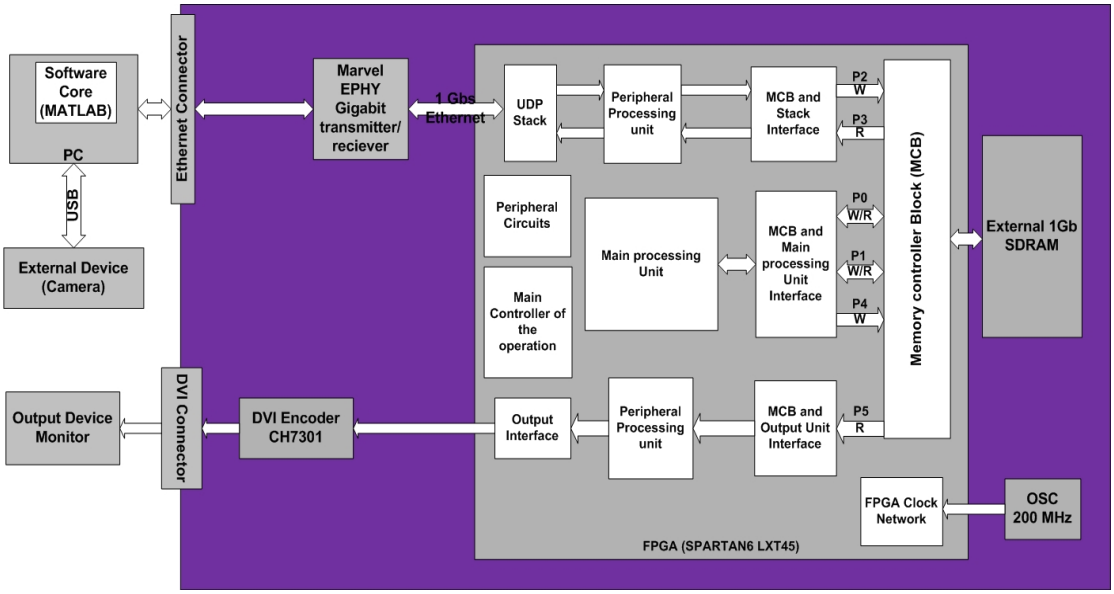

Fig.1. Overall Architecture of Proposed System

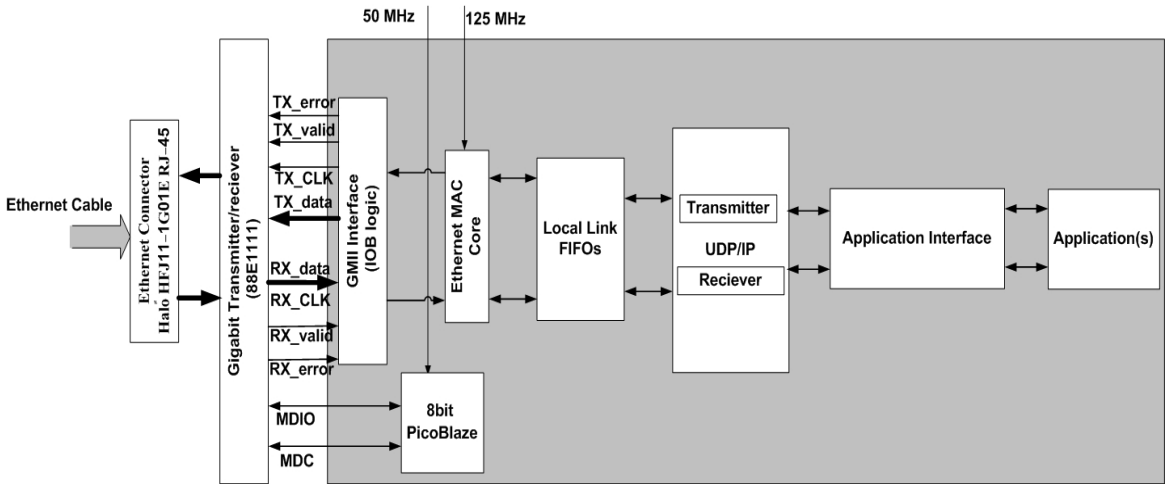

Fig.2. Overall Architecture of the UDP Stack

This layer is designed using TEMAC (Tri-mode Ethernet MAC) core, two FIFOs and other peripheral circuits. TEMAC is offered by Xilinx in order to take care of tasks such as framing, error detection, overflow control, padding, etc with high capabilities and in variety of operation modes. This core is configured to operate at full-duplex (which is faster than half-duplex and it consumes less FPGA slices) with minimum possible resource consumption and thoroughly suitable for data streaming applications.

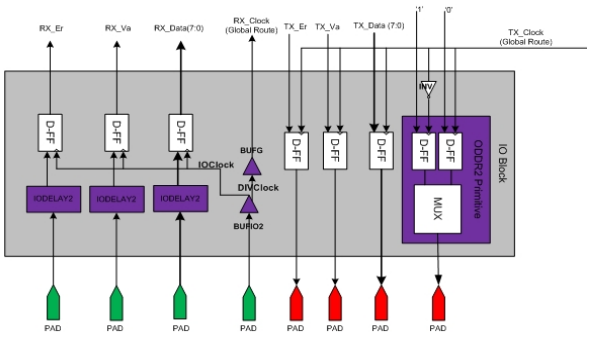

Fig.3. GMII Standard Interface Circuit in Spartan-6 IOBs

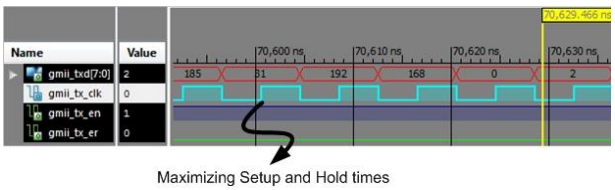

Fig.4. Timing Diagram of GMII-TX signals in FPGA IOBs

After receiving a full Ethernet frame, the related information is reported to upper layers regarding the condition of the incoming packet; in consequence, the decision will be taken more efficiently. The gigabit transmitter/receiver device is managed by an 8 bit Picoblaze microcontroller. Instead of using MDIO interface modules, the core is configured using 64-bit configuration vector in the UDP layer making the design more efficient.

Local link client interface is mainly consists of two parameterizable FIFOs; One handles receiving tasks and the other one handles sending packets. Each of the FIFOs is 4096 bytes long which means they can hold a complete standard Ethernet frame with maximum allowable length. The specific feature of these FIFOs is the simple protocol which is considered at both sides of them (client and TEMAC). Using several controlling signals, these FIFOs are capable of transferring the packets with arbitrary length and controlling the overflow (of consecutive packets). The transmit data is first buffered in the FIFO. When the TEMAC is ready to transmit that, the FIFO will be emptied; consequently, header, trailer and padding bytes if it was necessary are included. Also, the minimum interframe gap is considered when sending consecutive packets.

The UDP core comprises of two independent units, the transmitter and receiver units. The receiver unit, upon receiving an ARP request packet, informs the transmitter which sends a reply frame as soon as possible. The schematic view of the UDP layer is depicted in Fig. 5.The transmission is performed by TX controller. It controls the priority of the data and the type of ARP (request or reply, broadcast) which should be sent. RX controller deals with TX controller, application IF (interface), and Local Link FIFO. These controllers are mainly used to control the overall operation of the core. The ARP transmitter consists of an FSM, two simple 45 bytes distributed RAMs to store data, and some other basic logic blocks. This module is designed pipelined. ARP transmitter begins sending data when the output line is free to offload data, a request is asserted by TX controller, and local link TX FIFO is ready to accept data.

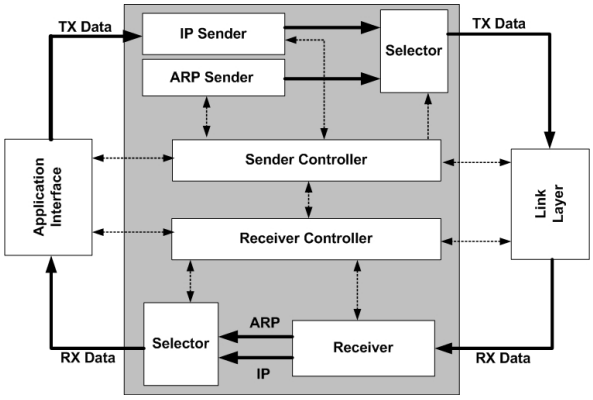

Fig.5. The Structure of the UDP Core

IP transmitter consists of a control unit to manage the whole data sending operation, i.e. to control a checksum calculator to calculate the UDP checksum, a BRAM FIFO to store the data while the checksum is calculated, a ROM to store headers, and finally a multiplexer to choose the data which should be transmitted, i.e. the payload, the header or the checksum bytes. User can simply modify the ROM and change the IP addresses, the length of the payload, etc. RAM is a simple dual port RAM implemented using a dedicated FPGA BRAM. It is large enough to hold 2048 bytes of data, considering that the length of an Ethernet frame is limited to 1538, with maximum payload size of 1500 bytes (1472 bytes of data, 20 bytes of IP header and 8 bytes of UDP header). BRAM is connected directly to the control unit which provides enable signals and the addressing during both “read” and “write” operations. The multiplexer works with the select signals that come from the control unit and the input signals from the BRAM, ROM and checksum register. In this scheme, the stream of data is downloaded to the local link FIFO accurately in accordance with the protocol. Each byte of the data is offloaded at the rising edge of the clock.

The controller block in the transmitter unit includes some sub-blocks, mainly an FSM in its heart that controls timings, input and output signals, addressing of BRAM and ROM, selecting proper signals for multiplexers and handshaking with upper and lower layers. It is a 7-state finite state machine with 16 transitions, 5 inputs, 9 outputs, and a synchronous reset. It is implemented by using FPGA LUTs based on one hot encoding. The transmitter unit is in idle state when there is no information to send.

When an application requests to send information, a state transition happens and the core starts buffering the incoming bytes. Meanwhile, the checksum unit calculates the incoming packet's checksum. When the whole data buffered, the controller inquires the local link interface to check if data transmission is possible, and by receiving acknowledgement, transmission will start in the next state. To do that, first header bytes and then payload will be placed into the TX FIFO of the local link interface. In this state, there is a possibility that another packet is on the line to be transmitted by a similar (or different) application. In this case, transmitter while sending data to the local link can accept the incoming payload since the BRAM is dual port. Since the length of the payload is fixed, transmission will be finished sooner than data receiving. Hence, controller waits till rest of the data from application arrives (to complete checksum calculation) and then it will start sending headers. Checksum is calculated whenever a segment should be transmitted. The design has high performance and speed, and this has been achieved by using FPGA dedicated blocks, employing high level of pipelining, breaking and reordering long critical paths to meet timing constraints, and also by using advance HDL coding techniques specific to Spartan-6 structure.

The receive unit similarly consists of several sub-modules and one is the controller. The controller manages all the incoming and outgoing signals of the unit. The data is sent out directly, but an indicator signal triggered by controller determines its validity. The controller contains an FSM, a couple of counters and buffers and some low level peripheral blocks. The FSM has six states. In its idle state, the circuit is waiting for trigger from lower layer to start transmission. In the next state, the MAC-add-buffering state, 14 bytes of sender and receiver MAC address and the type/length of protocol are checked. Appropriate reactions are done after that. The IP buffer is next state in which bytes of data in IP header or ARP header are received. The UDP buffer state is based on 8 bytes of UDP header. In statistic state, the results of the UDP checksum calculation and header comparisons are sent to the application. The application decides to ignore or use incoming bytes based on these validation signals.

The IPv4 checksum calculator operates only on the IPv4 header by start and finish signals from the controller. The UDP checksum calculator operates on pseudo IP header, UDP header and data. A simple small RAM (32 bytes) is considered in the design to store those bytes of the IP header that included in pseudo IP header. In controller, the output of the counter that count the number of incoming bytes of IP header is decoded to the addresses of the simple RAM to write the incoming bytes correctly into its cells. Soon after finishing data transmission, the checksum operation is performed with the bytes coming from the controller. The final checksum value will be calculated and based on this value, a warning signal will be sent to application by the controller.

B. MCB Interfaces

Memory controller block (MCB) is a dedicated embedded block; a multi-port memory controller that simplifies the task of interfacing FPGA devices to the most popular memory standards. The FPGA's hard memory controller is used for data transfer across the DDR3 memory interface's 16-bit data path using SSTL15 signaling. In the proposed architecture, the MCB is fed with a 400 MHz differential clock. In addition, 3 unidirectional and 2 bidirectional 32-bit ports configuration is established accompanying with an arbitration scheme which maximizes the performance. A PLL is employed to supply this block with the memory system clock and the calibration clock. The schematic of the MCB interface with other blocks is depicted in Fig. 6. Each module that aims to access the external RAM is required to burst a command. In each of the system memory clock cycle (i.e. 3000 Ps), MCB checks the availability of any command in each port's command FIFO based on the arbitration priority. Depend on the burst length of that command and FIFO statues, data will be transferred.

Considering an image processing application, Ethernet data bytes are merged to form 3-byte data words. The data received from UDP/IP unit is checked in terms of accuracy in IP addresses, MAC addresses, checksum, etc. If the data is correct, the 3-byte data word will be shifted into the other block accompanying with a valid signal. This valid signal is asserted 1 time in every 3 cycles. Of course, this scheme is not critical for all applications. In present design, each pixel consists of Red, Green and Blue bytes and each of these bytes arrive in order (i.e. pixel rate is 1 third of 125 MHz). In addition, the end of receiving a full frame of an image (considered 640*480) is also indicated in this part and used in controller to provide address updating.

MCB Interface blocks are units through which the memory controller is accessed. Two MCB Ethernet interface blocks are considered in the design scheme; one is to buffer the incoming Ethernet data packets into the external SDRAM and the other is to fetch data from external RAM and send it over Ethernet to the PC. The architecture is simple and efficient. Writing the incoming data directly into the DDR3 via MCB is quite inefficient since this may waste a significant portion of MCB's bandwidth; hence, when using the system in full operation, achieving real-time operation will be not practical. In order to avoid this, data which is fetched from Ethernet stack is first buffered in an interface FIFO which is specialized by its independent read and write clocks. The data is written in FIFO as soon as it received with exactly the frequency of incoming bytes (pixel rate); however, in order to read data, a relatively higher frequency should be chosen depend on the device characteristics, the design constraints mostly related to the PAR result and also the effective bandwidth which is needed. Whenever the number of pixels in the FIFO reaches a threshold, the process of vacating FIFO embarks on. Using this scheme, the maximum efficiency is obtained. Peripheral circuits are required for controlling the whole operation.

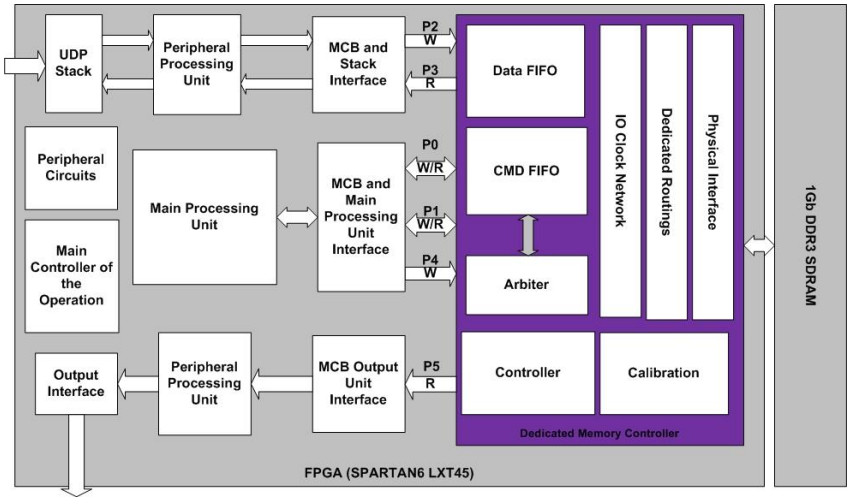

Fig.6. MCB and Connected Blocks in Proposed System

The Output-MCB interface block heavily depends on the application. If the DSP algorithm is an image processor, then this block should be an interface to monitor. Here, the output interface is based on two main units. One module is directly connected to the MCB and is responsible for reading processed pixels from the SDRAM while the other one consists of sub-circuits designed mainly for controlling monitor synchronization signals, communicating with the DVI encoder and handshaking with former module. The unit which is directly connected to the MCB is comprised of two FIFOs for storing image lines according to the protocol which is described in the following. An FSM sub-block controls the entire operation which is performed on 5 states. The idle state in which the system is waiting for the first image frame to be ready; the first line state in which the FSM waits so that the first line of image is stored in the odd FIFO; the 3rd and 4th states that will be consecutively repeated and switched whenever there is a change in lines; and the 5th state which is dedicated to the retrace time between two consecutive frames. The idea is that when one of FIFOs is receiving data from MCB, the other is sending the former line of image to the output. The write and read frequency for both FIFOs are independent and different. The read depends on the refresh rate of the output video (here is 25 MHz), and the write clock is much higher to compensate the MCB delay and to utilize its limited bandwidth efficiently.

C. Addressing and Memory Partitioning

Dedicated MCBs also simplify the task of addressing as the processing module could perform byte-byte addressing (similar to the SRAM based systems). In other words, MCB converts the input addresses into either Row-Bank-column or Bank-Row-Column which is mainly utilized in DDR memories. In order to manage the memory space, it is divided into non-overlapping regions; each of which is large enough to encompass a full frame of an image. Considering this, a controller is considered at the top level of the design which manages the addressing of each module. Regardless of the unit which is accessing the RAM, an address is divided into two parts; a dynamic part which is specified by the unit itself, and a static part which is determined by the controller. In fact, the static part specifies an image region (a division of RAM and it is constant for all of pixels of an image) and the dynamic part locates the exact location of the pixel in that image. There are three major modules with direct access to the external RAM; the Ethernet unit, image processing unit, and finally, output unit (monitor interface). Thus, in steady state of operation, one unit writes an image into the RAM while the other is processing the former frame and the last one is reading the last processed image.

When the FPGA is receiving the first frame of video, only the Ethernet interface writes the image into the RAM and the other two units are inactive. When the last pixel of the first frame was written, the image is ready in the SDRAM to be processed though receiving all of the pixels of a frame is not always necessary. Nevertheless, when the processing trigger is activated by the controller, the main image processing task begins. Similarly, when processing of one frame is over, the output interface reads the processed data and sends them to the monitor (shown in Fig. 7). The whole operation will continue until either the camera is disconnected or the device is turned off. In the first case, the system is designed in such a way that the last successfully processed frame will be shown in monitor (fixed). One important fact which should be considered is that the throughput of processing data must be equal or more than the throughput of storing the data into the DDR3 and the throughput of reading the processed data must be more than both of them so that loss of data will be zero.

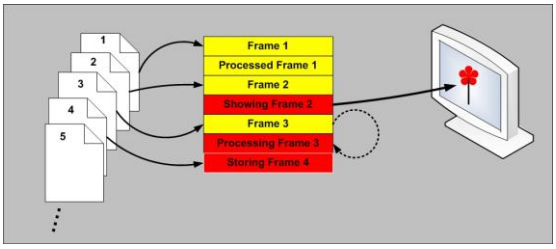

Fig.7. Memory Partitioning And Simultanous Storing, Processing and Reading Data

D. Clock Network Infrastructure

In the design of this system, various clock signals with frequencies in the range of 25 MHz up to 250 MHz are considered. The structure of the clocking infrastructure is depicted in Fig. 8. The IBUFGDS block is a clock buffer for differential input signals which is used here to generate a single 200 MHz signal in order to derive a DCM and a PLL. The DCM generates clock signals which are required in Ethernet unit (three 125 MHz signals; one not buffered and two buffered with opposite phases, a 200 MHz signal for MCB interface and a 25 MHz to supply the microcontroller which is connected to the Gigabit transceiver), the video logic (a 125 MHz signal for general logic circuitry, a 25 MHz signal to derive the monitor and its synchronization signals and a 200 MHz clock for the MCB interface), and finally the processing unit with a 200 MHz and a 125 MHz signals. In case, the main image processing requires higher or lower frequencies depending on its design specifications, additional DCM blocks are available for usage. A PLL is to generate required clock signals for the MCB; hence, the one with the nearest physical distance to the MCB is employed. Due to the fact that DCMs do not have access to the I/O clock network, it is not possible to use them. The due signal clocks are buffered in the physical layer of the interface and also, two additionally strobe signals (which are required by the MCB) are generated using BUFPLL primitive. For MCB calibration, a clock signal must be routed from the so-called PLL and in phase with clk400. A typical value for the frequency of this signal is 100 MHz.

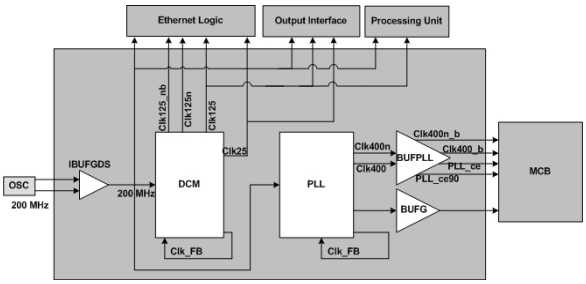

Fig.8. Clock Network Infrastructure

III. Experimental Setup and Results

In Section II, the overall architecture of this general system has been described. Considering this, practical implementation of variety of image processing applications is possible. In order to corroborate the validity of the system, two image processing algorithms namely a skin detection and a motion detection is designed. For the former, the aim is to group all the pixels into two classes of skin and non-skin pixels for any image [26, 27]. This has numerous applications in surveillance, content based coding, and face detection [28, 29, 30], etc. One specific method of skin segmentation called explicitly defined method [28] is implemented to observe the accuracy of the setup. In addition, a motion detection algorithm is also implemented based on frame differencing technique which can be used either as a standalone application or preprocessing for many image processing algorithms. MATLAB as a high level application provides raw data for the FPGA as well as receiving and processing the output data. In MATLAB, DSP system toolbox is used to send and receive UDP packets. Here, a network node is developed to communicate with FPGA. The software core also utilizes the connected camera in order to capture the video (24-bit color images) with the rate of 30 fps. Each frame of the video constitutes a number of Ethernet packets and then it is transmitted to the FPGA via the network node. In addition to the MATLAB, a network protocol analyzer, The Wireshark, is exploited in order to observe the network traffic in debug and verification phases. The operation starts when MCB calibration is finished. After executing initial necessary connection commands, PC sends an ARP request to the FPGA. Other data packets based on other protocols may be transmitted to the FPGA via other applications in PC; however, they will be filtered out in FPGA. Through image transmission process each frame of image (a single image or still video) will be sent to the FPGA for further processing.

Table 2. UDP/IP Utilization Ratio

-------	[38]	[23]**	[23]*	[22]	Proposed
Device	**Sp3**	**Sp3**	**Sp3**	**Sp3**	**Sp6**
Slices	**111**	**1022**	**517**	**184**	**123**
BRAM	**0**	**3**	**3**	**0**	**0**
DSP	**0**	**0**	**0**	**0**	**1**
Length	**1472**	**256**	**256**	**1472**	**1472**
ARP	**No**	**Yes**	**no**	**no**	**Yes**
F-max	**132**	**60.3**	**90.7**	**128.8**	**239.3**

Table 3. System's Resource Utilization Ratio

Block	Utilization without skin module	Utilization with skin module	Utilization with skin and motion module	Available on XCSL6LX45T
Slice Reg	**2,460**	**2,530**	**3325**	**54,576**
Slice LUT	**2,531**	**2,606**	**2876**	**27,288**
Bonded IOBs	**101**	**103**	**105**	**296**
RAMB16BWERs	**8**	**9**	**14**	**116**
RAMB8BWERs	**2**	**3**	**2**	**232**
DCM/DCM_CLKGENs	**1**	**1**	**2**	**8**
MCB	**1**	**1**	**1**	**2**
PLL_ADV	**1**	**1**	**1**	**4**
DSP48A1	**1**	**4**	**4**	**58**

Xilinx Integrated Software Environment, ISim and ChipScope analyzer were used for implementation, simulation and debugging purposes respectively. The resource utilization for UDP/IP stack is provided in Table. II and it is compared to that of previous works. Also, in [31-36], the design of a network stack is provided; however, they are incomparable with the present work as the design objectives are not the same. The resource utilization for the proposed method is based on the output of PAR. The resource utilization ratio of the whole system is provided in Table. III that shows the system consumes a low percentage of resources and that there is still sufficient amount of resources for implementation of variety of DSP algorithms. In addition, denser devices can be used for implementation of more resource demanding algorithms. Since data is dynamically transmitted through FPGA and high dense SDRAM, by efficiently designing DSP modules in FPGA, a small number of BRAMs will be required in implementation of many algorithms while preserving the speed needed in many applications. In Fig. 6, the result of the performing skin detection using both FPGA and MATLAB are compared for a single image. The result of skin segmentation for video is depicted in Fig. 7 while Fig. 8 represents the implementation of motion detector. In both of Fig. 10 and Fig. 11, the live video which comes from the camera is shown in the left monitor and the result of applying the algorithms is depicted on the other one. Finally, in Fig. 9, the routed design including both the skin detection and the motion detection modules are depicted.

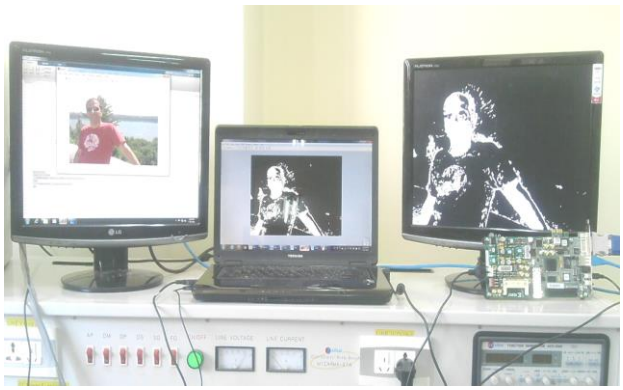

Fig.9. Experimental Setup(Left: Original, Middle: MATLAB Simulation, Right: FPGA Implementation)

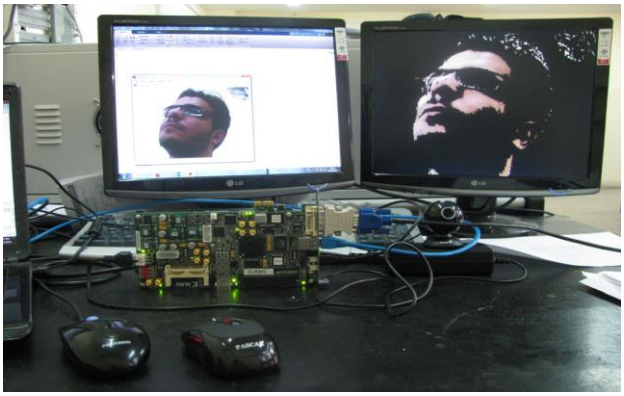

Fig.10. Implementation of a Skin Detection Algorithm

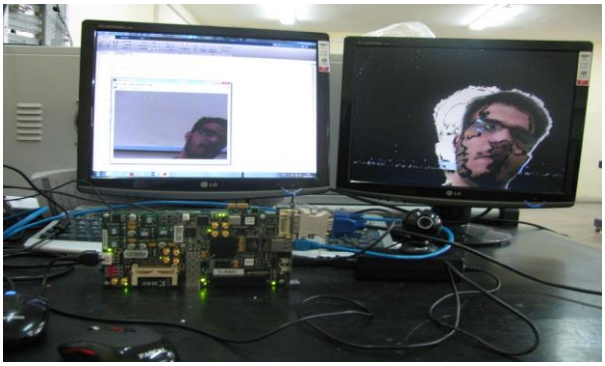

Fig.11. Implementation of a Motion Detection Algorithm

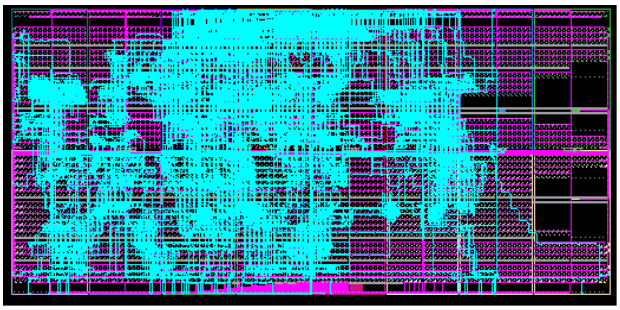

Fig.12. Routed Design

IV. Conclusion

DSP applications have been numerously developed in recent years, and MATALB is a strong tool in prototyping different algorithms with simple, cheap and fast design flow. However, in many practical systems, using general purpose computers will lead to poor performance. In this case, FPGAs are one of the best choices as the implementation platforms. In this paper, a new high speed and low cost FPGA-MATLAB interface is proposed that can be used in developing and prototyping many FPGA based DSP algorithms and it provides a remarkable solution to FPGA-MATLAB communication for variety of applications particularly image processing systems. The proposed system operates on a 1Gbs Ethernet link between PC and FPGA and it is based on UDP protocol. The incoming packets are stored in an off-the-shelf SDRAM and dynamic data transmission between FPGA and memory will be the rest of FPGA design. This proposed system consumes negligible amount of FPGA resources which can be very important in many applications. The entire system can be implemented using a low cost FPGA. The validity of the system was successfully confirmed by implementing a skin detection algorithm and a motion detection system.

References

[1] U. Meyer-Baese, “Digital signal processing with field programmable gate arrays,” Vol. 65. Heidelberg: Springer, 2007.

[2] R. Woods, J. McAllister, Y. Yi, and G. Lightbody, “FPGA-based Implementation of Signal Processing Systems”, John Wiley & Sons, 2008.

[3] J. Cooley, and T. Tukey, "An algorithm for the machine calculation of complex Fourier series," Mathematics of computation 19, no. 90 (1965): 297-301.
[4] C. D. Cunţan, I. Baciu, and M. Osaci. "Studies on the Necessity to Integrate the FPGA (Field Programmable Gate Array) Circuits in the Digital Electronics Lab Didactic Activity." International Journal of Modern Education & Computer Science 7.6 (2015).
[5] T. Daghooghi, "Design and Development MIPS Processor Based on a High Performance and Low Power Architecture on FPGA." International Journal of Modern Education and Computer Science (IJMECS) 5.5 (2013): 49.
[6] B. Rashidi, and M. Sabahi, "High Performance FPGA Based Digital Space Vector PWM Three Phase Voltage Source Inverter," International Journal of Modern Education and Computer Science (IJMECS) 5.1 (2013): 62.
[7] S. Kim, J. Park, S. Park, B. Koo, K. Shin, K. Suh, I. Kim, N. Eum, and K. Kim, "Hardware-software implementation of MPEG-4 video codec," ETRI journal 25, no. 6 (2003): 489-502.
[8] S. Ramachandran, "Digital VLSI systems design," springer, 2007.
[9] I. Kuon, and J. Rose, "Measuring the gap between FPGAs and ASICs," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 26, no. 2 (2007): 203-215.
[10] H. Parvez, and H. Mehrez, "Application-specific Mesh-based Heterogeneous FPGA Architectures," Springer, 2010.
[11] R. Francis, J. Rose, and Z. G. Vranesic, "Field-programmable gate arrays," Vol. 180. Springer, 1992.
[12] S. Haruyama, "FPGA in the Software Radio," IEEE communications Magazine (1999): 109.
[13] National Instruments, "Reconfigurable Grid? FPGAs Versus DSPs for Power Electronics," 2011.
[14] J. Rodriguez-Andina Mar á, J. Moure, and M. Dolores Valdes, "Features, design tools, and application domains of FPGAs," Industrial Electronics, IEEE Transactions on 54, no. 4, pp.1810-1823, 2007.
[15] R. Tessier, and W. Burleson, "Reconfigurable computing for digital signal processing: A survey." Journal of VLSI signal processing systems for signal, image and video technology 28, no. 1-2, pp. 7-27, 2001.
[16] A. Banerjee, A. Sundar Dhar, and S. Banerjee, "FPGA realization of a CORDIC based FFT processor for biomedical signal processing," Microprocessors and Microsystems 25, no. 3, pp. 131-142, 2001.
[17] M. R. Mahmoodi, H. Nikaein, Z. Fahimi, "A Parallel Architecture for High Speed BLAST Using FPGA," In Electrical Engineering (ICEE), 2014 International Conference on, IEEE, 2014. In press.
[18] Fahimi, Z.; Dehghani, R., "IM3 injection technique for improving the linearity of an RF power amplifier," in Electrical Engineering (ICEE), 2015 23rd Iranian Conference on, vol., no., pp.1112-1117, 10-14 May 2015 doi: 10.1109/IranianCEE.2015.7146379L.
[19] Guan, and A. Zhu, "Low-cost FPGA implementation of Volterra series-based digital predistorter for RF power amplifiers," IEEE Transactions on Microwave Theory and Techniques 58, no. 4, 2010.
[20] X. Tan, S. Chen, Z. Zhou, and F. Zhang, "Face recognition from a single image per person: A survey," Pattern Recognition 39, no. 9 (2006): 1725-1745.
[21] S. Kasap, K. Benkrid, and Y. Liu, "Design and Implementation of an FPGA-based Core for Gapped BLAST Sequence Alignment with the Two-Hit Method," Engineering Letters 16, no. 3 (2008): 443-452.
[22] N. Alachiotis, S. Berger, A. Stamatakis, "A versatile UDP/IP based PC ↔ FPGA communication platform," Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on , vol., no., pp.1,6, 5-7 Dec. 2012
[23] A. Lofgren, L. Lodesten, S. Sjoholm, and H. Hansson, "An analysis of FPGA-based UDP/IP stack parallelism for embedded Ethernet connectivity," In NORCHIP Conference, 2005. 23rd, pp. 94-97. IEEE, 2005.
[24] M. R. Mahmoodi, S. M. Sayedi, and B. Mahmoodi, "Reconfigurable hardware implementation of gigabit UDP/IP stack based on spartan-6 FPGA," Information Technology and Electrical Engineering (ICITEE), 2014 6th International Conference on. IEEE, 2014.
[25] K. Ross, and J. Kurose, "Computer Networking: A Top-Down Approach Featuring the Internet: Preliminary Edition," Addison-Wesley Longman Publishing Co., Inc., 1999.
[26] M. R. Mahmoodi, S. M. Sayedi, "Boosting performance of face detection using an efficient skin detection algorithm," In Information Technology and Electrical Engineering (ICITEE), 2014 International Conference on, IEEE, 2014.
[27] M. R. Mahmoodi, S. M. Sayedi, "A Face detection method based on kernel probability map", Computers and Electrical Engineering, Elsevier, 2015.
[28] M. R. Mahmoodi, S. M. Sayedi, and F. Karimi, "Propagation from conservatively selected skin pixels using a multi-step multi-feature method," Electrical Engineering (ICEE), 2015 23rd Iranian Conference on. IEEE, 2015.
[29] M. R. Mahmoodi, S. M. Sayedi, "A face detector based on color and texture." In Information Technology and Electrical Engineering (ICITEE), 2014 6th International Conference on 2014 Oct 7 (pp. 1-6). IEEE.
[30] M. R. Mahmoodi, and S. M. Sayedi, "Leveraging Sp 10 Analysis on Homogeneous regions of Color Images for Skin Classification," Computer and Knowledge Engineering (ICCKE), 7th International Conference on, pp.1-6, IEEE, 29-30 Oct, 2014.
[31] F. Herrmann, et al, "A gigabit udp/ip network stack in fpga," Electronics, Circuits, and Systems, 2009. ICECS 2009. 16th IEEE International Conference on. IEEE, 2009.
[32] A. Dollas, et al. "An Open TCP/IP Core for Reconfigurable Logic," FCCM. Vol. 5. 2005.
[33] W. Kühn, et al. "FPGA based compute nodes for high level triggering in PANDA," Journal of Physics: Conference Series. Vol. 119. No. 2. IOP Publishing, 2008.
[34] P. Bomel, G. Gogniat, and J. Diguet, "A networked, lightweight and partially reconfigurable platform," Reconfigurable Computing: Architectures, Tools and Applications. Springer Berlin Heidelberg, 2008. 318-323.
[35] Xilinx, "XAPP433: Embedded System Example: Web Server Design Using MicroBlaze Soft Processor," 2006.
[36] N. Alachiotis, S. A. Berger, and A. Stamatakis, "Efficient PC-FPGA Communication over Gigabit Ethernet," Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on. IEEE, 2010.

Authors' Profiles

Mohammad Reza Mahmoodi was born in Isfahan, Iran, in 1989. He received his B.Sc. and M.Sc. degree in Electronics from Isfahan University of Technology (IUT) in 2011. Since then, he is pursuing his Ph.D. in Electronics at UCSB. His areas of interest include analog/digital circuit design, computer vision, pattern recognition and image processing.

Sayed Masoud Sayedi was born in Maragheh, Iran, in 1960. He received the B.Sc. and M.Sc. degrees in electrical engineering from Isfahan University of Technology (IUT), and the Ph.D. degree in electronics from Concordia University in1986, 1988, and 1996, respectively. From 1988 to 1992, and then since1997, he has been with IUT, where he is currently an associate professor in the Department of Electrical and Computer Engineering. His areas of interest include VLSI fabrication processes, low power VLSI circuits, and data converters.

