FPGA accelerator for real-time skin segmentation

Bart de Ruijsscher, Georgi N. Gaydadjiev
Computer Engineering Laboratory
Electrical Engineering, Mathematics
and Computer Science Department
Delft University of Technology
Mekelweg 4, 2628CD, Delft
the Netherlands
bderuijsscher@gmail.com
g.n.gaydadjiev@ewi.tudelft.nl

Jeroen Lichtenauer, Emile Hendriks
Information and Communication Theory Group
Electrical Engineering, Mathematics
and Computer Science Department
Delft University of Technology
Mekelweg 4, 2628CD, Delft
the Netherlands
J.F.Lichtenauer@ewi.tudelft.nl
E.A.Hendriks@ewi.tudelft.nl

Abstract

Many real-time image processing applications are confronted with performance limitations when implemented in software. The skin segmentation algorithm utilized in hand gesture recognition as developed by the ICT department of Delft University of Technology presents an example of such an application. This paper presents the design of an FPGA based accelerator which alleviates the host PC's computational effort required for real-time skin segmentation. We show that our design utilizes no more than 88% of the resources available within the targeted XC2VP30 device. In addition, the proposed approach is highly portable and not limited to the considered real-time image processing algorithm only.

1 Introduction

The field programmable gate arrays (FPGAs) capability to implement application-specific computations is widely used to improve algorithm performance. Reconfigurable computing is a new paradigm of the last fifteen years that considers hybrid computing machines composed by a general purpose processor and an FPGA [11, 15, 17]. Such custom computing machines have been shown very efficient for steaming data applications such as multimedia [6, 14].

Skin segmentation is a multimedia operation commonly utilized in applications such as hand gesture recognition and behavior monitoring. It refers to the classification of image areas which have a high probability of containing a skin color. Several different classification methods exist, however these methods often rely on calibration of the camera in order to provide for accurate results. The method proposed in [9] comprises a novel approach to skin detection without requiring camera calibration. The skin segmentation algorithm applies an adaptive linear 2D projection of RGB values to extract illumination invariant color information, computes the mahalanobis distance to the skin color mean and applies a strict threshold on this result. A more forgiving threshold is applied to pixels in the vicinity of groups of the first detected skin pixels, by means of local counting and binary dilation. Finally, a binary closing is applied to cover missing skin pixels. Furthermore, in order to provide higher algorithm layers with a measure of motion estimation, the skin segmentation algorithm also calculates the absolute difference between two consecutive frames. The software implementation of this method has proven to be limited in performance when utilizing an input video resolution of 320x240 or higher and a frame rate of 20 frames per second. Software profiling indicated that the majority of the CPU time is consumed by the pixel operations such as color space conversions and morphological operations such as erosion and dilation [13].

This paper focuses on the design of an FPGA accelerator for real-time skin segmentation. It effectively implements all skin segmentation algorithm stages in hardware and is organized in accordance to its software implementation. Our accelerator is designed to meet the requirements of real-time hand gesture recognition applications. The main contributions of this paper are:

- true real-time acceleration of the skin segmentation algorithm;
- highly portable design due to the used standard network interface;
- generally applicable infrastructure suitable for acceleration of other real-time image processing algorithms.

The remainder of this paper is organized as follows. Section 2 provides a short overview of previously reported related work. Then section 3 discusses the general architecture of the system. Section 4 presents the architecture of the *pixel processing pipeline* (PPP). Evaluation of the system will be discussed in section 5. Finally, the discussion is concluded in section 6.

2 Related Work

Traditionally, application specific ICs have been employed for acceleration of image processing operations. The main drawback of such systems is their inflexibility. Reconfigurable hardware provides a good alternative for application specific hardware without compromising on flexibility.

Several PCI based implementations of reconfigurable hardware coprocessors implementing real-time image processing operations have been reported. For example, [19] proposes an architecture based on specialized image processing modules which can be loaded in the FPGA configuration. A different approach is presented in [16] and proposes a coprocessor specifically designed to accelerate the addressing of individual pixels within software applications.

Several authors have proposed a high-level approach to the problem of implementing image processing operations in reconfigurable hardware. For example, [12] proposes a method of describing image processing operations using single assignment C (SA-C). Such a description can be converted in a dataflow graph by a specialized compiler and implemented in reconfigurable hardware. Another high level method was proposed in [4] and essentially comprises a dedicated compiler that uses predefined FPGA configurations.

Architectures designed for specific applications have also been discussed in the literature. For example, an architecture for vision based navigation based on log polar transformation is proposed in [2]. An FPGA implementation of a pixel processor for object detection applications is discussed in [10] and the implementation of 2-D feature detection on an FPGA is presented in [1].

In respect to the above related work our design differs in the following. Instead of a PCI interface we employ an Ethernet communication channel between a host processor and our accelerator. This allows our system to be used with a wide range of host PC's (both desktops and laptops). The high-level approaches and the specific FPGA implementations as described above can be applied to the pixel processing pipeline used in our proposal. Said this our approach provides a more general scenario.

3 Proposed Architecture

Our system consists of a host PC and an external FPGA accelerator. The communication link between both systems is a 10/100 MBps direct ethernet connection. The input images are captured by a webcam connected to the PC and split into pixel packets on the host PC. Then, the pixel packets are transferred to the accelerator which processes the packets and returns the results.

Considering this connection type and the fact that very high bandwidth utilization is essential for the overall system performance, we adopted UDP for our communication protocol. There are two packet types: data (also referred to as pixel packets) that carry consecutive pixel- or accelerator output information and control packets used for system configuration at run-time. The pixel packets are 1358 bytes long and contain a single image line with 4 bytes per pixel (320*4 = 1280 pixel bytes). The remaining 78 bytes are used for the following headers: accelerator (38), UDP (8), IP (18) and MAC (14 bytes). The accelerator header contains an image frame identification number, x and y coordinates of the first pixel present in the packet and a camera identification number. The camera identification number can be used in a multiple camera system setup. The accelerator output consists of 4 bytes per pixel which contain the absolute difference, grey value and skin segmented output.

The control packets have the same size as the pixel packets in order to simplify the protocol handling on the PowerPC processor present in the FPGA. The meaningful part of the payload is only 80 bytes. This is, however, not a serious issue considering that control packets are usually not intermingled with pixel data at run-time.

The architecture of the accelerator is depicted in figure 1 and consists of several distinct components which are interconnected by the system bus. The following components are present:

Ethernet media access layer component (EMAC): the EMAC is responsible for receiving and sending of data packets. When an incoming packet is received, it interrupts the processor in order to transfer the packet to the data memory.

Processor: the processor is responsible for the overall coordination of the components present. Furthermore, it must ensure a timely data transfer between the different components. The processor transfers incoming and outgoing packets from and to the EMAC, and it moves data to and from the pixel processing pipeline (PPP) subsystem.

Data memory: this memory is used as an intermediate buffer between the ethernet component, the processor and the pixel processing pipeline subsystem.

Instruction memory: this memory contains the program code (instructions) for the processor.

Pixel processing pipeline subsystem: the PPP is responsi-

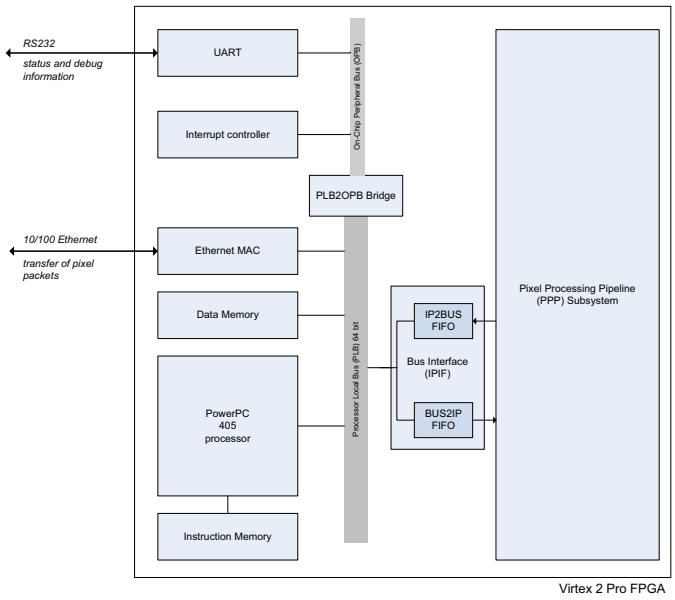

Figure 1. System architecture

ble for performing the skin segmentation algorithm as discussed in [9].

PPP Bus Interface (IPIF): provides bus protocol handling in order to simplify the design of the PPP subsystem. It also provides two additional FIFO's for temporal storage.

The interrupt controller, UART, bus bridge and system busses are standard building blocks provided by Xilinx.

The interaction of the different components directly involved in processing pixel packets can be illustrated by the following example of a receive packet event:

1. an incoming pixel packet arrives at the EMAC;
2. the EMAC receives the packet in an internal buffer and generates a processor interrupt;
3. the processor starts an interrupt handling routine which transfers the packet into the data memory and analyzes the packet header;
4. the processor transfers the packet to the pixel processing pipeline input buffer (BUS2IP);
5. the PPP processes the packet on a pixel by pixel basis and stores the results in the PPP output buffer (IP2BUS), and generates an interrupt when a complete pixel packet has been processed;
6. the processor starts the interrupt routine which transfers the packet back into its data memory;
7. the processor transfers the packet to the EMAC and orders the EMAC to transmit the packet to the host PC;
8. the EMAC transmits the packet to the host PC using the direct Ethernet connection.

The responsibilities for the processor furthermore include initialization of the entire system and the network protocol handling required for the peer to peer connection with the host PC. Furthermore, it allows run-time reconfiguration of different algorithm parameters by using the configuration packets.

Communication Bandwidth Evaluation: The available network bandwidth in the proposed infrastructure is envisioned to form a performance bottleneck and thus to have an impact on the maximal image resolution that can be used. The following bandwidth requirement estimation is considered:

$$BW = depth_{bits\_per\_pixel} * I_w * I_h * R_{frame} * F_{overhead}$$

,where the $depth$ indicates the amount of bits required to represent a single pixel, I_w (width) and I_h (height) indicate the image dimensions, R indicates the frame rate and F represents the overhead factor caused by the additional header data present in a pixel packet. In our system, the pixel packets contain 1280 bytes of pixel data and 78 additional bytes required for the protocol headers. This results in approximately 6% overhead.

Assuming a pixel depth of 32 bits, a frame rate of 20 frames per second and an overhead factor of 6%, the resulting bandwidth requirements are 49,69 Mbps for image dimensions of 320x240, 198,75 Mbps for 640x480 and 508,80 Mbps for 1024x768 pixels. Since our implementation utilizes a 10/100 MBps Ethernet MAC, it is limited to image dimensions of 320x240 pixels, however a gigabit Ethernet MAC would allow for image dimensions of 640x480 or 1024x768.

4 Pixel Processing Pipeline

The pixel processing pipeline is responsible for the operations performed on the incoming pixels. Since we require parallelism at the level of elementary algorithm operations, we decided to implement the PPP by utilizing a pixel pipelined architecture.

The original software implementation of the skin segmentation algorithm can be divided into several dependent stages. Each stage contains a certain amount of different independent operations which can be executed simultaneously. Figure 2 shows the architecture of the pixel processing pipeline. The operations indicated correspond to those mentioned in the discussion of the hardware implementation of the skin segmentation algorithm [5]. The dotted lines between the pipeline stages represent pipeline registers which contain the temporary results from the preceding stage. The pipeline controller is responsible for the coordination of the entire processing pipeline. It determines

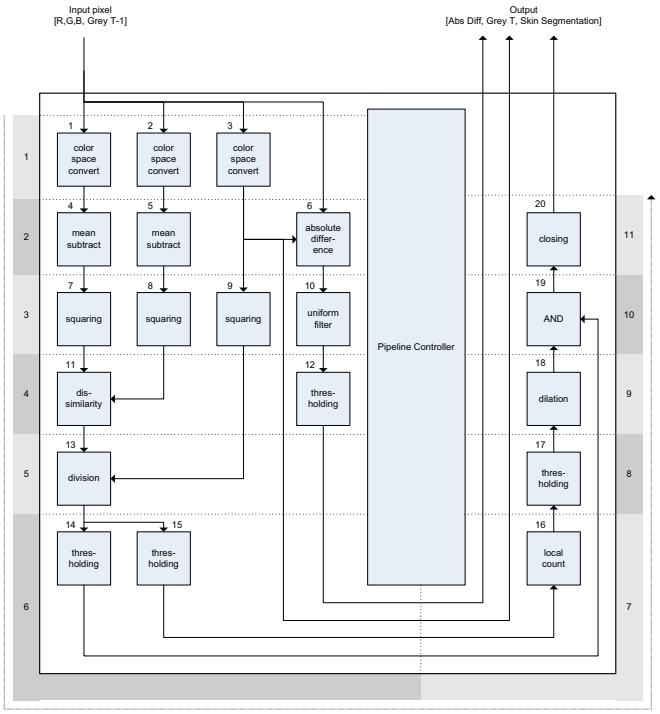

Figure 2. Pixel processing pipeline

Table 1. XC2VP30 utilization summary

	used	total	ratio
Slices	12,141	13,696	88%
Slice registers	11,045	27,392	40%
4-input LUTs	17,459	27,392	63%
Block RAM	124	136	91%

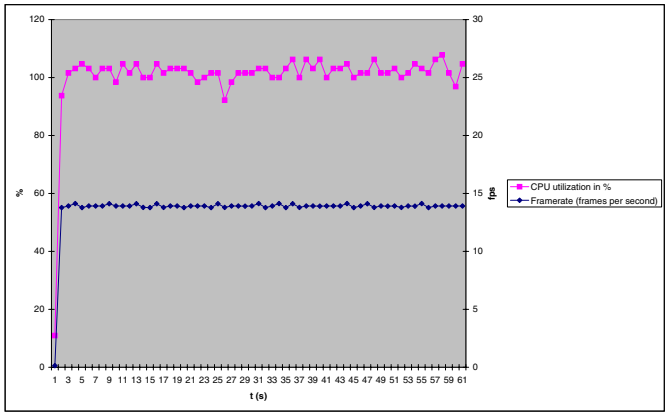

Figure 3. Software algorithm performance

when all pipeline stages have finished processing in order to initiate the next pipeline cycle. More precisely, we have implemented and tested the following image processing operations: color space conversion (vector-matrix multiplication), basic motion estimation (absolute difference calculation), 5x5 greyscale uniform smoothing filter, binary local count filter and binary erosion, dilation and closing. The remaining components presented in figure 2 (e.g. division and squaring) have been implemented by simply utilizing available mathematical components provided by the Xilinx IP library.

5 Prototype Evaluation

For our evaluation study we used a stand alone workstation based on a 2,8 GHz Pentium 4 processor with 1024MB memory running Windows XP Pro as our reference and measurement system. The system setup additionally consists of a Philips ToUCam Pro webcam.

The accelerator prototype has been implemented on a Virtex 2 Pro FPGA using the XUPV2P development board [21]. A summary of the resource utilization is shown in table 1. The total utilization of the logic resources (slices) is 88% while only 37% are required for the implementation of the pixel processing pipeline. The remaining resources are used by the accelerator architecture. The high utilization of block RAM is due to the instruction and data memory space needed for the PowerPC processor.

First, the performance of the algorithm software implementation was evaluated. Figure 3 shows both the CPU utilization and the achieved frame rate. As can be observed from the measurements, the software version running on the host workstation is unable to achieve the real-time performance criterion of 20 frames per second. A maximum frame rate of 14 fps can be achieved at a CPU utilization close to 100%.

Figure 4 shows the results of our system prototype utilizing the proposed hardware accelerator. The CPU utilization rate varies between 60% to 70% while the frame rate remains steady around 20 fps. This is a considerable improvement compared to the software version since the CPU utilization rate is significantly lower while achieving the required frame rate.

The frame rate speedup gained by our hardware accelerated implementation is $20/14 \approx 1.43x$. Moreover, the CPU utilization is 30% to 40% lower compared to the software implementation. Thus it allows enough processor resources for higher level applications such as hand gesture recognition.

6 Conclusions and future work

In this paper we have presented an FPGA based accelerator for real-time skin segmentation of streaming 320x240, 20fps video data. We showed that the video stream dimensions are limited only by the available bandwidth of the Eth-

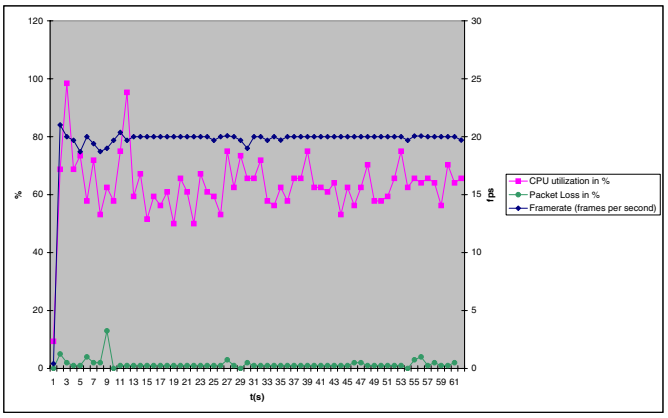

Figure 4. Hardware implementation performance

ernet link used to transfer the data between the workstation and our accelerator. Considering the fact that all specific operations are implemented solely in the pixel processing pipeline, the same general architecture can be employed for accelerating different pixel based image processing operations.

Future work on this accelerator should concentrate on reducing the packet loss and the communication overhead. Several optimizations related to network protocol handling (as mentioned in amongst others [3], [7] and [8]) can be considered to reduce the communication overhead currently present in our demonstrator. Furthermore, the design could also be implemented using a gigabit ethernet enabled platform [20] in order to support input video resolutions of 640x480 pixels and higher. In addition, implementation of the proposed pixel processing pipeline as custom computing unit in MOLEN [18] will allow avoiding the data transfer bandwidth bottleneck of the current system.

References

[1] A. Benedetti and P. Perona. Real-time 2-d feature detection on a reconfigurable computer. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1998.

[2] J. Boluda, F. Pardo, F. Blasco, and J. Pelechano. A pipelined reconfigurable architecture for visual-based navigation. *Proceedings of the25th EUROMICRO Conference*, 1999.

[3] J. Chase, A. Gallatin, and K. Yocum. End system optimizations for high-speed tcp. *IEEE Communications Magazine*, 39:68–74, 2001.

[4] D. Crookes, K. Benkrid, A. Bouridane, K. Alotaibi, and A. Benkrid. Design and implementation of a high level programming environmentfor fpga-based image processing. *IEE Proceedings Vision, Image and Signal Processing*, 147:377–384, 2000.

[5] B. de Ruijsscher. Fpga based accelerator for real-time skin segmentation. Msc. thesis, Delft University of Technology, Mekelweg 4, June 2006. CE-MS-2006-08.

[6] S. Dutta and W. Wolf. A flexible parallel architecture adapted to block-matching motion-estimation algorithms. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(1):74–86, February 1996.

[7] A. Foong, T. Huff, H. Hum, J. Patwardhan, and G. Regnier. Tcp performance re-visited. *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, 2003.

[8] J.-H. Huang and C.-W. Chen. On performance measurements of tcp/ip and its device driver. *Proceedings of the 17th Conference on Local Computer Networks*, pages 586–575, 1992.

[9] J. Lichtenauer, E. Hendriks, and M. Reinders. A calibrationless skin color model. *Proceedings of the Twelvth annual conference of the Advanced School for Computing and Imaging*, June 14-16, 2006.

[10] P. Mc Curry, F. Morgan, and L. Kilmartin. Xilinx fpga implementation of a pixel processor for object detection applications. *Celoxica application note*, 2001.

[11] M.Wazlowski, L.Agarwal, T.Lee, A.Smith, E.Lam, H.Silverman, and S.Ghosh. PRISM-II Compiler and Architecture. In *Proc.IEEE Workshop on FPGAs for Custom Computing Machines*, pages 9–16, Napa Valley,CA, April 5-7, 1993.

[12] W. Najjar, B. Draper, A. Bhm, and R. Beveridge. The cameron project: High-level programming of image processing applications on reconfigurable computing machines. *Proceedings of the Workshop on Reconfigurable Computing (PACT), Paris*, 1998.

[13] Rafael C. Gonzales and Richard E. Woods. *Digital Image Processing*. Prentice Hall, second edition, 2002.

[14] S.C.Goldstein, H. Schmit, M.Moe, M.Budiu, S.Cadambi, R.R.Taylor, and R.Laufer. PipeRench: A coprocessor for Streaming Multimedia Acceleration. In *26th International Symposium on Computer Architecture*, pages 28–39, Atlanta, Georgia, May 1999.

[15] S.M.Trimberger. *Reprogramable Instruction Set Accelerator*. U.S. Patent No. 5,737,631, April 1998.

[16] W. Stechele, L. Alvado Crcel, S. Herrmann, and J. Lidn Simn. A coprocessor for accelerating visual information processing. *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, 2005.

[17] S. Vassiliadis, S. Wong, and S. D. Cotofana. The molen $\rho\mu$-coded processor. In *in 11th International Conference on Field-Programmable Logic and Applications (FPL), Springer-Verlag Lecture Notes in Computer Science (LNCS) Vol. 2147*, pages 275–285, August 2001.

[18] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. Panainte. The molen polymorphic processor. *IEEE Transactions on Computers*, 53, 2004.

[19] M. A. Vega-Rodrguez, J. M. Snchez-Prez, and J. A. Gmez-Pulido. Real time image processing with reconfigurable hardware. *Proceedings of the 8th IEEE International Conference on Electronics, Circuits and Systems, 2001. ICECS 2001.*, pages 213–216, 2001.

[20] Xilinx System Engineering Group. Gigabit system reference design. *Xilinx Application Note XAPP536*, 2004.

[21] Xilinx System Engineering Group. *Xilinx University Program Virtex-II Pro Development System*, 2005.