W R C

Workshop on **R**econfigurable **C**omputing

Workshop Proceedings

January 28, 2007

Sofitel, Ghent, Belgium

An FPGA-based System for Development of Real-time Embedded Vision Applications

Hongying Meng[1], Nick Pears[1]
Chris Bailey[1]

Department of Computer Science, The University of York, York, YO10, 5DD, UK

ABSTRACT

In this paper, a FPGA based system for real-time video processing is proposed. In this system, real-time video was obtained though a digital camera and video/image processing algorithms were implemented on FPGA hardware. The original image frames and processed video can be displayed simultaneously on the screen of a PC, acting as a system monitor. It is a flexible and easily implemented system, which is used as a quick and simple platform for FPGA-based video processing algorithm development and evaluation. Furthermore it is a cheap solution especially for low complexity real-time video processing development for applications on small, low-power devices such as mobile or ubiquitous devices.

KEYWORDS: FPGA; video processing; computer vision; real time

1 Introduction

Due to the rapid development of the video camera and mobile video, video processing has attracted more and more attention in recent years. Real-time video processing is among the most demanding computation tasks [Atha95]. One solution is to use special purpose DSP processors, designed to execute certain low-level image processing operations extremely quickly. However, this approach can sometimes be over specialized, when one has a large range of possible applications in mind, since this requires a commensurately large range of low and medium level operations. An alternative solution to a software implementation (running on either general or specialized hardware) is the design of specific hardware for specific computer vision processes, in order to perform a high rate of operations per second. If only a small number of specific processes are required for a particular application, then only those particular processes need to be implemented in the hardware.

Continuing growth in silicon chip capability is rapidly reducing the number of chips in a typical system, and increasing the size, performance and power benefits of System-on-Chip (SoC) integration. However, the design and test of a miniaturized vision system can be quite strenuous, owing to many technology and financial constraints that often restrict the developer's pool of resources.

[1]Email: {hongying,nep,chrisb}@cs.york.ac.uk

Advances in programmable logic devices have resulted in the development of Field Programmable Gate Arrays (FPGA) that allow integration of large numbers of programmable logic elements in a single chip. The size and speed of FPGAs are comparable to ASICs, but FPGAs are more flexible and their design cycle is shorter. It is possible that FPGA architectures will allow generic real-time image processing, computer vision and pattern recognition techniques to be packaged with a relatively low power CPU and an image sensor.

Actually, FPGA has been used in computer vision system frequently [Schm04] [Sold99] [McCr00] [Fern05] [Masr06]. Schmittler et al. [Schm04] used a single FPGA chip for real-time ray tracking of dynamic scenes. Soldek and Mantiuk [Sold99] proposed a FPGA based system for pattern recognition, processing, analysis and synthesis of images. McCready and Rose [McCr00] used FPGA based system for real-time, frame-rate face detection. Vargas-Martín et al [Fern05] proposed a generic FPGA based real-time video processing system for applications in the field of electronic visual aids. Masrani and MacLean [Masr06] used FPGA for a real-time large disparity range stereo system.

Based on the wide applications of FPGA in the area of computer vision, some people have tried to develop a system or design methodology for this kind of designs [Flei98] [Dray99] [Hayn00] [Arri02] [Sen05]. Fleischmann et al. [Flei98] proposed a hardware/software prototyping environment for networked embedded systems. Drayer and Araman [Dray99] proposed a development system for creating real-time machine vision hardware design on FPGAs. Haynes et al. [Hayn00] proposed the Sonic architecture that is a configurable computing system performing real-time video image processing. Arribas and Macia [Arri02] proposed a FPGA based system for development and testing of vision algorithms such as image compression and optical flow calculation. Sen et al. [Sen05] developed a design methodology for generating efficient target hardware description language code from an algorithm.

Increasingly, there are some commercial FPGA development boards such as Xilinx Virtex-4 Video Starter Kit available with a video I/O solution. However, lots of them were designed for a specific chip or algorithms, for example, DSP based algorithms. Meanwhile, the price of these commercial boards is also high. In most cases, such as intelligent surveillance, traffic management or object detection in mobile video, a very simple low-cost FPGA development board is all that is needed.

Although above great progress, it is still helpful to have a cheap and convenient tool to develop computer vision algorithm for FPGA chips. In this paper, a novel PC-FPGA video development platform was provided. It was designed for quick development of ubiquitous computing applications with a demand for real-time video processing. The rest of this paper is organized as follows: In section 2, we will give a brief introduction to the architecture. In section 3, Huffman coding will be explained. In section 4, FPGA design of the components are presented .In section 5, experimental results are shown. Finally, we present the conclusions.

2 System architecture

Figure 1 shows the basic outline architecture of the system for the development of real time video applications. It is a very cheap solution for FPGA based video processing algorithm development. A PC is connected with a digital camera by USB cable while it is also connected with a FPGA board. The PC can grabs the video frames from the camera and when the FPGA board is ready, the frames can be directly sent to FPGA board and be processed. Meanwhile, the results can be sent back and

displayed on the screen of PC along with the original frame. Obviously in a real ubiquitous/embedded application, we would remove the PC and connect the camera directly to the FPGA board, which directly implements the appropriate camera driver logic.

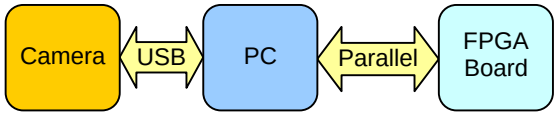

Figure 1: Architecture of the PC-FPGA System

2.1 Hardware construction

From figure 1, it can be clearly seen that this is an easily implemented system. There are plenty of FPGA boards suitable for this system. For the FPGA development boards, some of them have a parallel connection with PC while others may have an USB, PCI or other high-speed connection. However, some of these are rather expensive. A simple FPGA development board is needed here to connect to PC by parallel cable. This parallel cable can be used for downloading FPGA design from PC to FPGA board. It can be switched to transfer image signals between PC and FPGA board.

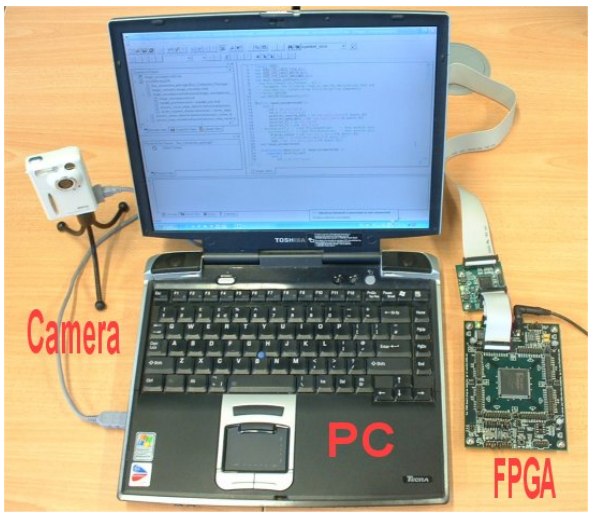

Figure 2: Our PC-FPGA System

In our experiment, a simple BurchED FPGA prototyping board (BX-300 FPGA) was selected. There is a Xilinx SPARTAN 2E chip with 300K gates on this board. It is connected to a PC by a parallel cable through a JTAG connector. FPGA design can be downloaded to the FPGA board through the parallel cable.

For digital camera, lots of digital camera or web camera can be used here. In our system, the DC1300 from BenQ was used. It has a USB connection to PC. BenQ DC1300 has many functions such as capture digital still image, image frames and record video slip. But here, it was used only as a web camera. Figure 2 showed our cheap PC-FPGA system.

2.2 Software components

The software components include the ones both on PC side and FPGA sides, as described in figure 3. On the PC side, we have developed a friendly interface using Visual C++. The original video or image sequences were captured and transferred to PC by the software though a USB connection. The software was designed in VC++ environment. It can capture the video frames and display them on the screen. Huffman algorithm codes were designed to compress and decompress the images. There are also some other codes such as image format transformation between RGB and YUV.

The software components on the PC side consist of an operating-system windowing/message processing layer, image acquisition, a parallel port interface and image display. A basic WIN32 message loop handles things like startup and shutdown and passes on repaint and input calls to the main code. The image acquisition code either supplies images from files or captures data from a device such as a digital camera. This uses the Microsoft Vision SDK API and assumes a Video For Windows compatible device has been set up. The image display code uses OpenGL. The PC-side code also does Huffman encoding and decoding of the images, as described earlier. Meanwhile, there is a module designed for parallel port interface using the DLPortIO library.

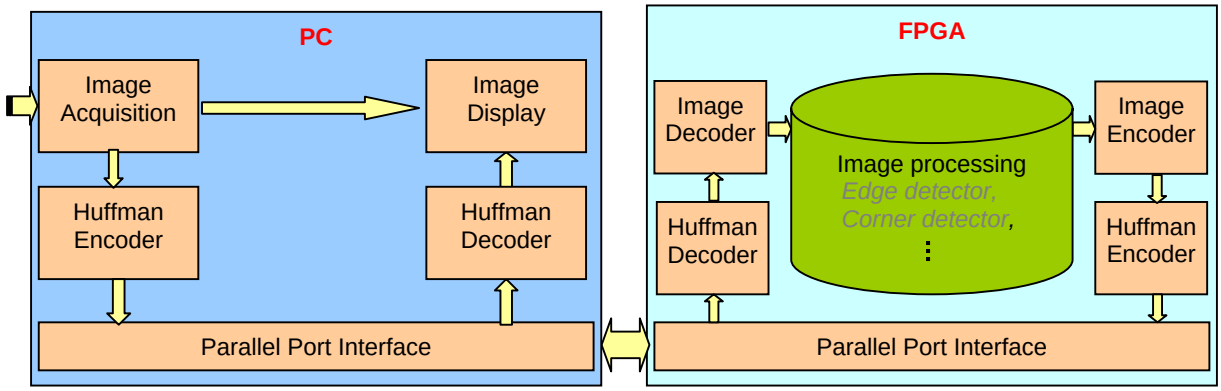

Figure 3: Software components in the system

On the FPGA side, six modules were designed for the FPGA chip. They are parallel port interface, Huffman decoder, Huffman encoder, image decoder, image encoder and image processing modules. The image processing module is the key element, which we wish to develop, whereas the other elements are essentially 'test harness' components of the system for image acquisition, display and evaluation of FPGA-based image processing algorithm performance. This image processing algorithm can be any image processing algorithm required by the application. For each module, VHDL codes were synthesized and implemented by Xilinx tools in ISE 7.1i (WebPACK) and then they were downloaded to the FPGA chip SPARTAN 2E using iMPACT tool. Finally, the parallel cable was switched to data communication mode and the FPGA board reset. When the software on PC side starts to run, the real-time video can be processed and displayed on the PC.

3 Huffman coding design

Huffman coding [Huff52] is a well-known entropy encoding algorithm used for lossless data compression. The term refers to the use of a variable-length code table for encoding a source

symbol (such as a character in a file) where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol. Instead of determining symbol frequencies, the code table used here is fixed and used every frame. We used a few sample images to determine typical symbol frequencies.

The image is first transformed into 3 channels: the brightness and two colour channels. The brightness is stored to a higher accuracy than the colour channels because it is more noticeable to the eye.

Next, each pixel $p_{i,j}$ is examined and it's represented as the difference $d_{i,j}$ from its neighboring pixels:

$$d_{i,j} = p_{i,j} - (p_{i-1,j} + p_{i,j-1})/2 \quad (1)$$

That way, large areas of similar brightness/colour are represented as low values.

The final step is to encode this data with the Huffman code table. There are actually two code tables: one for the brightness channel and another for the colour channels.

There are two different code tables - one for encoding and one for decoding. The encoder is just a lookup table where it just transforms one fixed-width symbol to one variable-width symbol. The decoder needs to analyse a stream of bits and delineate the symbol boundaries then translate the variable-width symbols into fixed-width ones.

When the bit stream is read, the matching prefix is found in this table and the symbol length is examined - that many bits are then removed from the stream and translated to a fixed-width (un-encoded) symbol. Then the reverse of the original image translation is applied to pixel differences and combining the colour channels.

4 FPGA components design

For our current system, five basic FPGA modules have been designed, as shown in figure 3. The parallel port interface module was used for data communication with PC. Image encoder was used to convert the image format from RGB to YUV, while the image decoder did the reverse process. The Huffman encoder and decoder were used for compression and decompression.

Table 1. The sizes of FPGA components.

Components	**Slices**	**%**
The Parallel port interface	14	0
Huffman decoder	168	5
Image decoder	306	9
Sobel edge detector algorithm	189	6
SUSAN edge detector algorithm	848	27
SUSAN corner detector algorithm	1010	32
Image encoder	34	1
Huffman encoder	78	2

The image processing algorithms that have been constructed are Sobel edge detection [Gonz92] and SUSAN edge and corner detection [Smit95]. The sizes of these components, when synthesized, are listed in table 1.

Pipelining was introduced to components that had lower maximum clock speeds, as this is the limiting factor for the maximum clock speed of the design as a whole. When the design occupies more chips are then the maximum as a whole. The system clock speed can become significantly lower than the slowest component (if it were to be synthesized on it's own). This is due to the longer signal pathways, which can represent more than half the delay, compared to the actual delay from the logic itself. By segmenting the component and pipelining, a speedup can be achieved for very little area cost, at the expense of increased latency time between input and output.

5 Experimental results

Figure 4 demonstrates an experimental result obtained from our system. Real-time video frames were captured through the camera and, in this example, the camera was fixed and the hand was moving. The resultant images of edges extracted by the Sobel edge detector are displayed at the same time as the real-time video. The sampling rate of the frame was estimated and controlled automatically by the software itself. In our experiments, 5 frames per second rate can be reached, which is a limit due to the bandwidth of the parallel port, rather than a limit due to the image processing algorithm itself. Currently we are developing more FPGA-based processing modules using this development system in order to track people through video sequences.

Figure 4: Screen shot of real-time video edge extraction with successive 12 frames in which the camera was fixed and the hand was moving.

6 Conclusions

In this paper, a FPGA based real-time video processing development system was introduced. This system can be used for development and evaluation of real-time video processing algorithms on FPGA chips and is especially useful for developing prototypes for mobile and ubiquitous, visually-driven applications. It is an easily implemented and extensible system with large flexibility and low cost.

7 Acknowledgements

The authors would like to thank DTI and Broadcom Ltd. for the financial support for this research. We also would like to thank Mr. Peter Stock for his previous work on this project.

References

[Atha95] P.M. Athanas and A.L. Abbott, Real-time image processing on a custom computing platform, In *IEEE Computer*, Feb. 1995.

[Arri02] P. Arribas and F. Macia, FPGA board for real time vision development systems, In *Fourth International Caracas Conference on Devices, Circuits and Systems*, Aruba, Dutch Caribbean, 2002.

[Dray99] T. Drayer and P. Araman, A Development System for Creating Real-time Machine Vision Hardware using Field Programmable Gate Arrays. In *32nd Annual Hawaii International Conference on System Sciences (HICSS-32)*, Maui, Hawaii, 1999.

[Fern05] Fernando Vargas-Martín, M. D. Peláez Coca, Eduardo Ros, Javier Diaz, Sonia Mota. A generic real-time video processing unit for low vision. In *International Congress Series*, Vol. 1282, pages 1075-1079, 2005.

[Flei98] J. Fleischmann, K. Buchenrieder and R. Kress, A hardware/software prototyping environment for dynamically reconfigurable embedded systems. In *Proceedings of the Sixth International Workshop on Hardware/Software Codesign*, CODES, Seattle, Washington, USA, pages 105-109, 1998.

[Gonz92] R. Gonzalez and R. Woods, Digital image processing, *Addison Wesley*, pages 414–428, 1992.

[Hayn00] S. D. Haynes, J. Stone, P. Cheung, W. Luk, Video Image Processing with the Sonic Architecture, *IEEE Computer*, vol. 33, no. 4, pages 50-57, 2000.

[Huff52] D.A. Huffman, A Method for the Construction of Minimum-Redundancy Codes, In *Proceedings of IRE*, pages 1098-1101, 1952.

[Masr06] D. Masrani and W. MacLean, A Real-Time Large Disparity Range Stereo-System Using FPGAs. In *Lecture Notes in Computer Science*, 3852, pages 42-51, Springer 2006.

[McCr00] R. McCready and J. Rose, Real-time, frame-rate face detection on a configurable hardware system, In *FPGA '00: Proceedings of the 2000 ACM/SIGDA eighth international symposium on Field programmable gate arrays*, page 221, New York, NY, USA, 2000. ACM Press.

[Schm04] J. Schmittler, S. Woop, D. Wagner, W. J. Paul and P. Slusallek, Realtime ray tracing of dynamic scenes on an FPGA chip. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*. ACM Press, New York, NY, pages 95-106. 2004.

[Sen05] M. Sen, I. Corretjer, F. Haim, S. Saha, J. Schlessman, S. Bhattacharyya, and W. Wolf, Computer Vision on FPGAs: Design Methodology and its Application to Gesture

Recognition, In *The First IEEE Workshop on Embedded Computer Vision*, San Diego, USA, 2005

[Smit95] S. Smith and J. Brady, SUSAN – A new approach to low level image processing, In *Technical Report*, Oxford Centre for Functional Magnetic Resonance Image of the Brain (FMRIB), 1995.

[Sold99] J. Soldek and R. Mantiuk, A reconfigurable processor based on FPGAs for pattern recognition, processing, analysis and synthesis of images. In *Pattern Recognition Letters*, 20(7), pages 667-674, 1999.