A Reconfigurable Engine for Real-Time Video Processing

W. Luk, P. Andreou, A. Derbyshire, F. Dupont-De-Dinechin, J. Rice, N. Shirazi and D. Siganos

Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ, UK

Abstract. We describe the hardware and software extensions that transform a PC-based low-cost FPGA system into a reconfigurable engine for real-time video processing. The hardware extensions include a daughter board for the FPGA system which handles analog and digital colour video conversion. The software extensions include reusable libraries, development tools and a run-time environment. Applications include linear and non-linear filtering, edge detection, image rotation, histogram equalisation, colour identification, motion tracking, and creation of video effects. Our system has been used for research involving video processing, run-time reconfigurable circuits, and hardware/software co-design.

1 Introduction

Real-time video is becoming increasingly popular with the proliferation of low-cost video cameras, camcorders and other facilities. Real-time video processing, however, is among the most demanding computation tasks [1]. Application-specific integrated circuits can deliver implementations optimal in speed and size, but they are often inflexible, expensive and time-consuming to develop. Digital signal processing chips and general-purpose microprocessors often include hardware support for processing video or multimedia data, but there may be drawbacks. First, such hardware support may not cover a particular user-specific task; second, the system architecture and low-level software, such as interrupt control, can cause significant delay if not carefully optimised.

This paper describes the hardware and software extensions which transform an FPGA-based board into a reconfigurable engine for real-time video processing. The board we use has a user-reconfigurable Xilinx 6200 FPGA and two megabytes of static memory, and is available from multiple vendors for around 1000 US dollars. The key aspects of our work include: (a) the development of hardware and software components for a flexible, powerful and low-cost video processing engine, and (b) the use of techniques such as run-time reconfiguration and hardware/software codesign for optimising high-performance designs. A variety of application examples will be used to illustrate our approach.

Our work is inspired particularly by the VTSplash project [1] at Virginia Tech, which enhanced a multi-FPGA system hosted in a SPARC-workstation with monochrome video facility. Other FPGA-based video processing environments are based on the VME system [3], the transputer framework [11], or

FPGAs without partial reconfigurability [4]. Our system is PC-based, supports partial FPGA reconfiguration at run time, and can deal with colour video.

2 Framework

All of our designs for the video engine fit into the framework shown in Figure 1, which serves as a guide for structuring systems dealing with high-speed data streams. There are hardware elements PRE and POST for pre-processing and post-processing such streams. In the context of this paper, the PRE element accepts a video stream as input, performs the required processing such as subsampling or feature extraction, and passes the result to POST. The POST element performs further processing, such as labelling the features identified by PRE, before producing the output video stream. The mapping of hardware components into PRE and POST may not be unique, and they are often arranged in the form of a pipeline [1].

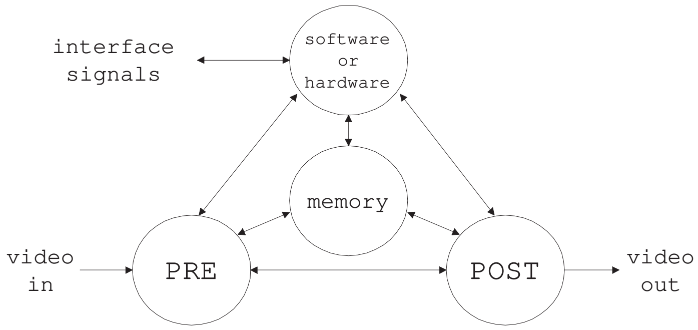

Fig. 1. A framework for structuring video designs, where PRE and POST are hardware elements for pre-processing and post-processing data. The hardware and the software components can be situated on the same or different boards.

Both PRE and POST may communicate with the memory or other hardware or software components. Reusable interface libraries have been developed to facilitate implementations involving the interactions between these components. A common situation is for PRE to extract features in the incoming video which can be described efficiently, since the feature descriptions can then be passed to a software procedure for further analysis using a low bandwidth link. Control information, such as configuration data, may also be sent from the software component to the hardware elements.

Run-time reconfiguration can be applied to this framework in several ways. First, the PRE and POST elements may be connected to video interface chips, and we will explain in Section 4 how reconfiguration techniques can be used to move seamlessly between changing the settings of these chips and processing the video data. Second, run-time reconfiguration can be used for optimising design size and speed by supplying a customised implementation at the appropriate time. In particular, there are techniques for effective reconfiguration of pipeline structures [9].

The scheme in Figure 1 has been shown to be useful not only for experiments involving video processing and run-time FPGA reconfiguration, but also for research in hardware/software partitioning and co-design. Software components are often better suited for some forms of processing than hardware components, for instance when irregular, data-dependent or floating-point computations are involved. The key to effective co-processing is to identify techniques and applications such that: (a) the hardware and software components perform tasks which match their capabilities, and (b) the communications between them do not overload the hardware/software interface.

As an example, we have developed a motion tracker which has a hardware component extracting motion information from an image and sending it to a software tracking procedure. The hardware component performs mainly regular and parallel computations at high speed, while the software component performs mainly irregular, sequential and floating-point operations less suitable for hardware implementation. Since for each video frame only a small amount of data are passed between the hardware and the software, the bus connecting them does not become a bottleneck. Further details of this application, and several others, will be discussed in Section 5.

3 Hardware

Our aim is to minimise efforts on system development, while the result should be flexible, low-cost and sufficiently powerful for real-time video operations. Flexibility and cost considerations motivate the use of a cheap but expandable FPGA-based board, preferably one capable of run-time reconfiguration. Performance consideration motivates specialised hardware support for video data transfer and for common operations such as hue, saturation and brightness control.

The approach that we adopt involves a low-cost development system based on the Xilinx 6200 FPGA, and a daughter board that we designed for interfacing the FPGA to real-time video sources and sinks (Figure 2). The hardware for the development system is a board which can contain either a Xilinx 6216 or a Xilinx 6264 device, on which the PRE and POST elements in Figure 1 can be implemented. The board has four 8-bit wide memories organised into two banks, and each bank can be accessed from either of the two address busses [10].

The FPGA board is interfaced to the PC through the PCI bus, but the PCI bus has often been found to be the bottleneck for real-time transfer of large image data [12]. To overcome this problem, we built a daughter board for converting between analog composite video and its digital version. The digitised video is sent to the FPGA board through a mezzanine connector. After processing, the result can be passed back to the daughter board for conversion into composite

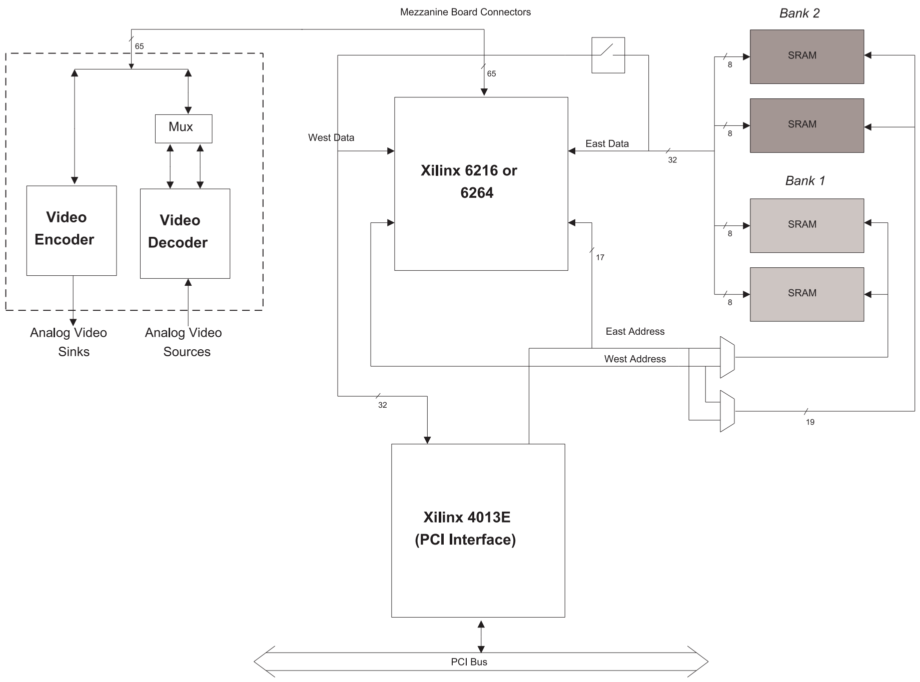

Fig. 2. Xilinx 6200 PCI system. The multiplexor `Mux` selects either the control ports or the data ports to connect to the mezzanine connector.

video for display. The display speed is only limited by processing delay, since we avoid using the PCI bus for transfer of uncompressed video. In essence, this method achieves high performance by eliminating system bus saturation and by shortcutting operating system delays.

The daughter board contains a video decoder and a video encoder, both from Brooktree Corporation. The decoder converts analog video, such as PAL or NTSC signals, to various digital formats, such as 24-bit or 16-bit RGB or YCrCb; the encoder performs digital to analog conversion. These chips also support other functions, including brightness and colour control. Since the number of lines on the mezzanine connector is limited, a scheme has been devised to share the control and data signals to the video decoder using a multiplexor (Figure 2). This method supports 24-bit colour input and output at the expense of losing video data whenever the settings of the decoder are adjusted.

4 Software

The software that has been developed for our reconfigurable engine include: (a) support for controlling the video interface chips, (b) memory interfaces,

(c) arithmetic and other building blocks in the form of parametrised libraries, (d) interface to software components, and (e) infrastructure for developing user applications involving real-time video. There are also compilation tools supporting run-time reconfigurable designs, and visualisation tools for debugging implementations.

The video interface chips described in the preceding section are powerful but require low-level programming. The programming involves specific data transfer protocols (such as I^2C and MPU) with complex timing constraints, which should be hidden from the designer of a video processing application. We adopt an approach which splits a video operation into two distinct phases. The initialisation phase involves setting up the FPGA to program the video interface chips through FPGA registers, with the multiplexor in Figure 2 selecting the control ports of the video decoder. A software library has been developed which transparently loads in an FPGA programming file defining all the required registers and their wiring to the relevant FPGA pins. There are high-level functions facilitating the configuration of the video interface chips to select the picture format, the colour depth and image intensity.

In the video processing phase, the registers linked to the programming of the video interface chips are frozen. The user design can access the video stream through standard interface components in various hardware description languages. The transition between the initialisation phase and the video processing phase is achieved by partially reconfiguring the Xilinx 6200 FPGA; this improves speed and minimises effects of transients on the video interface chips. The partial reconfiguration step is part of our framework for developing run-time reconfigurable systems, and is supported by various tools such as ConfigDiff [8].

Other components dedicated to the video daughter board include a memory interface capable of performing a read and a write cycle to the memory in each pixel clock cycle, and an address generator which produces the coordinates of the current pixel on the screen based on various clock and sync signals. Facilities are also available to enable the memory to be used as a double buffer, by using video synchronisation signals and a state machine generating the address and read/write signals. In the case of a line buffer, the current pixels are written to one memory bank while the pixels on the previous line are read from the other. This is reversed after each line, so a read and a write can be performed simultaneously. This method is less demanding on timing than attempting to read and write to the same SRAM bank in one cycle. The memory interface components have been demonstrated on various applications to be described in the next section.

Tools for application development consist of a compile-time environment and a run-time environment. The compile-time environment consists mainly of the development tools [8] and reusable libraries [5] for FPGA design. Hardware descriptions can either be expressed in VHDL or in the Pebble language [6], which can be regarded as a simple variant of VHDL used particularly by those without much background in electronics. Over 30 parametrised libraries have been developed in Pebble and VHDL, including various kinds of adders, multipliers and memory access circuits with different tradeoffs in processing speed and size. These libraries are also compatible with our tools for automating the production of run-time reconfigurable designs.

The run-time environment includes software libraries for monitoring and controlling the FPGA and the daughter board from the PC host. A graphical user interface has also been developed for design visualisation, debugging and demonstration. In all, the software extensions provide an infrastructure for rapid design implementation and for design reuse.

5 Applications

A variety of applications have been developed using this reconfigurable engine, including various forms of linear and non-linear filtering, edge detection, image transformation, histogram equalisation, colour identification, motion tracking and creation of video effects, all of which run at real-time rates.

One-dimensional operations. A variety of one-dimensional operators have been developed for our video processing system. These operators include binomial and median filters, and Sobel edge detectors. Often software is used in changing various settings by writing to registers on the FPGA, or in controlling run-time reconfiguration where customised circuits are downloaded onto the FPGA at the appropriate instants [7]. Memory can be used in transposing a video frame so that a two-dimensional operation can be obtained from composing two one-dimensional operations [2].

Colour detection. This application is used to detect a user-specified colour in the incoming video. The implementation filters out the other colours of the spectrum and only allows the detected colour to pass through. The processing is performed at a resolution of 640 by 480 pixels using 24-bit RGB colour.

Two different approaches were attempted. The first approach specifies the target colour by a range of values for each colour component. The detection is performed using six 8-bit comparators computing the maximum and minimum values for each colour component. Their outputs are combined to produce a signal representing the presence of the target colour. This design is sensitive to lighting conditions, since the criteria for colour identification are absolute.

The second approach identifies the target colour by the differences between the three colour components. For each pair of colours, the difference is calculated and then a comparator checks whether the difference is greater or smaller than a specified colour threshold. This method allows for more tolerance in brightness fluctuation, although further improvements can be achieved using another colour system such as HSV. The designs are reconfigurable at run time to enable the detection of a new target colour.

Image rotation. This application takes a video image and rotates it by an angle θ. This is accomplished by multiplying the co-ordinates of the incoming image by the formulae $x' = a \times x + b \times y$ and $y' = c \times x + d \times y$, where the coefficients $a = d = cos\,\theta$ and $b = -c = sin\,\theta$. Other transformations can be obtained by different coefficient formulae. Our design, which can complete four multiplications in two cycles, consists of two multipliers (Figure 3). The value of y is constant for the duration of each line; hence the products $y \times sin\,\theta$ and $y \times cos\,\theta$ can be performed at the blanking interval between lines, and in the remaining time the multipliers calculate the products involving x. The processing is performed on a 24-bit colour image at a resolution of 320 by 480 pixels; full resolution can be achieved by pipelining the multipliers. The encoder and decoder circuits convert coordinates to memory addresses and vice versa.

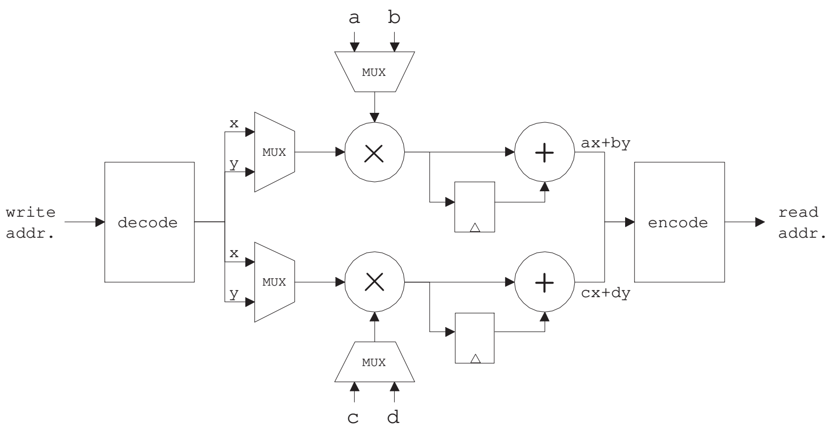

Fig. 3. Matrix multplier for image rotation design.

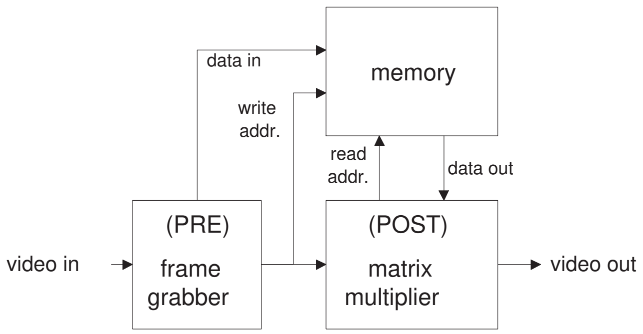

Fig. 4. System organisation for image rotation design.

The overall organisation is shown in Figure 4. The pre-processing step consists of optionally subsampling the image using a frame grabber and storing the result to memory. The matrix multiplier described above transforms the write address signals to read address signals, which can be used to extract a rotated image from memory and send it to the video output.

Histogram equalisation. Histogram equalisation has the effect of improving image contrast. Our design for histogram equalisation is similar to that for image rotation (Figure 4). The alterations include: (a) in addition to the frame grabber, the pre-processing step also involves generating a lookup table in memory which corresponds to a transformation function derived from the intensity histogram of the grabbed image; (b) the post-processing step consists of using the lookup table to transform the incoming video stream. Some calculations in the pre-processing step can be performed in software. This method can also be used to implement other computations involving lookup tables.

Edge detection. The edge detector described here is an example of a two-dimensional operator capable of processing full-rate, greyscale video derived from averaging the colour components. The gradient components of an image are computed by difference operators in the x and y direction. The difference operators are word-pipelined at every stage to keep the combinational delay within the pixel clock period. An edge is marked if the x or y gradient is above a given threshold; the thresholds are held in programmable registers and can be altered by software.

Video effects. Several examples involving video effects have been developed. We describe here an application to detect, in the video stream covering a sports event, a sign indicating an advertisement which will be replaced by another advertisement more appropriate for the local viewers. The task is complicated by the motion and zooming of the camera, and also by objects, such as the sportsmen, partly obscuring the sign.

The general solution to this problem requires processing power which is out of reach of the hardware described in this paper. However, if we assume that the camera rotates but does not translate, the position and size of the sign in the picture can be deduced from the angles and zoom values of the camera. From this information the software computes, for each video frame, the region holding the sign and writes this information to the FPGA for post-processing.

The POST component compares, for each pixel of the region containing the advertisement sign, its colour and the colour of the corresponding pixel in the initial advertisement scaled appropriately. The two pixels will be the same if nothing is obstructing the advertisement sign, in which case the pixel of the first advertisement is replaced by the corresponding pixel in the second. Otherwise there are objects in front of the sign which must be preserved in the image, so the incoming pixel will be transferred to the output.

Other applications have shown that appropriate scaling and restriction of the search space of the advertisement sign is straightforward. The main challenge is to decide whether two pixels have the same colour. We found that simple checks, such as comparing each colour in a pixel to within a certain range of the same colour in the corresponding pixel, are often too sensitive to lighting conditions.

We are currently exploring various colour representations and local algorithms, which may be able to produce a better result.

Motion tracking. Our motion tracker contains a hardware part and a software part. The PRE component of the hardware part consists of a differencer and a counter array; the POST component labels the detected moving objects in the output video stream. The software part contains a Kalman filter to minimise the effect of noise on the tracker performance (Figure 5).

A simple way of identifying motion is to compute the difference of the corresponding pixels in two consecutive video frames. The result is then thresholded to give a binary value for each pixel. The differenced frame is divided into blocks corresponding to different regions of the image. The amount of motion in each block can then be recorded by a counter; for instance an array of 64 counters will be required for a design with 8 by 8 blocks. A counter selector keeps a record of the current position of the incoming pixels on the screen, and enables the appropriate counter to increment accordingly.

Several FPGA-based implementations of motion trackers have been reported ([3],[11]). The advantages of our counter-based design include simplicity, flexibility and scalability. For instance, this method enables each pixel to be processed independently of the others within one cycle, so no inter-frame processing is necessary. Our method is also scalable: the availability of more hardware allows a larger counter array, and will usually lead to better performance.

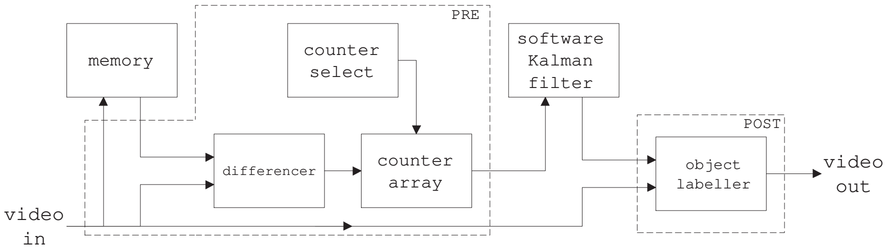

Fig. 5. A design for motion tracking.

The main function of the PC is to calculate the statistically 'best' estimate of the position and size of the object. Kalman filtering provides a method for making this estimate. It considers the statistical models of signal and noise to give optimum response to the actual motion, while giving minimum response to the noise present in its detection. Our Kalman filter involves mainly floating-point operations, and is best implemented in software. The post-processing on the FPGA consists of hardware for labelling the detected objects in the video stream, the locations of which are supplied by the PC.

6 Conclusion

Despite its simplicity and low cost, our reconfigurable engine has proved to be an effective vehicle for experimenting with run-time reconfiguration and hardware/software codesign techniques for video applications. We have shown that the PC can be used as the host to the video engine, and as a co-processor for executing complex, data-dependent and floating-point operations. Current and future work includes exploring further multimedia and other applications using our system, refining our framework and tools, and retargeting them to cater for other FPGA-based platforms.

Acknowledgements

Thanks to Stuart Nisbet for help in developing the video interface board. The support of Xilinx, Interval Research, Hewlett Packard Laboratories Bristol, INRIA, and the UK Engineering and Physical Sciences Research Council (Grant Number GR/L24366, GR/L54356 and GR/L59658) is gratefully acknowledged.

References

1. P.M. Athanas and A.L. Abbott, "Real-time image processing on a custom computing platform", *IEEE Computer*, February 1995.
2. M. Aubury and W. Luk, "Binomial filters", *Journal of VLSI Signal Processing*, vol. 12, 1996.
3. P. Dunn, "A configurable logic processor for machine vision", in *Field Programmable Logic and Applications*, W. Moore and W. Luk (editors), LNCS 975, Springer, 1995.
4. F. Lisa, F. Cuadrado, D. Rexachs and J. Carrabina, "A reconfigurable coprocessor for a PCI-based real-time computer vision system", in *Field-Programmable Logic, Smart Applications, New Paradigms and Compilers*, LNCS 1142, Springer, 1996.
5. W. Luk, S. Guo, N. Shirazi and N. Zhuang, "A framework for developing parametrised FPGA libraries", in *Field-Programmable Logic, Smart Applications, New Paradigms and Compilers*, LNCS 1142, Springer, 1996.
6. W. Luk and S. McKeever, "Pebble: a language for parametrised and reconfigurable hardware design", this volume.
7. W. Luk, N. Shirazi and P.Y.K. Cheung, "Modelling and optimising run-time reconfigurable systems", in *Proc. FCCM96*, IEEE Computer Society Press, 1996.
8. W. Luk, N. Shirazi and P.Y.K. Cheung, "Compilation tools for run-time reconfigurable designs", in *Proc. FCCM97*, IEEE Computer Society Press, 1997.
9. W. Luk, N. Shirazi, S. Guo and P.Y.K. Cheung, "Pipeline morphing and virtual pipelines", in *Field Programmable Logic and Applications*, LNCS 1304, Springer, 1997.
10. S. Nisbet and S.A. Guccione, "The XC6200DS development system", in *Field Programmable Logic and Applications*, LNCS 1304, Springer, 1997.
11. I. Page, "Constructing hardware-software systems from a single description", *Journal of VLSI Signal Processing*, Vol. 12, 1996.
12. S. Singh and R. Slous, "Accelerating Adobe Photoshop with reconfigurable logic", in *Proc. FCCM98*, IEEE Computer Society Press, 1998.