Implementing Photoshop™ Filters in Virtex™

S. Ludwig, R. Slous and S. Singh

Implementing PhotoShop™ Filters in Virtex™

Stefan Ludwig[1], Robert Slous[2] and Satnam Singh[2]

[1]Compaq Systems Research Center, Palo Alto, California, U.S.A.
Stefan.Ludwig@compaq.com
[2]Xilinx Inc., San Jose, California, U.S.A.
{Robert.Slous, Satnam.Singh}@xilinx.com

Abstract. This paper presents a complete system that utilises a FPGA-based co-processor to accelerate compute intensive image processing operations. Its main contributions are a methodology for incorporating hardware-based acceleration into a commercial image processing application by exploiting a plug-in architecture; a presentation of a new PCI-based FPGA accelerator system suited for image processing style applications; and theoretical calculations and empirical measurements of the system that was actually built.

1 Introduction

The design, implementation and performance analysis of a FPGA-based co-processor system for accelerating the image processing application Adobe Photoshop is presented. We describe a general purpose FPGA co-processor system using the Xilinx Virtex FPGA. We show how circuits performing various image processing applications are realised on this FPGA hardware. The software interface between the card and the Photoshop application is described as well as a description of how Adobe Photoshop was made to communicate with the FPGA hardware. We instrument the performance of software and hardware versions of two filters and compare against the theoretical performance of our hardware platform.

2 Image Processing with Adobe Photoshop

Adobe markets a series of applications for producing or processing drawings (Adobe Illustrator), photographic quality pictures (Adobe Photoshop) and video (Adobe Premier). In the paper we shall concentrate on the acceleration of Adobe Photoshop, but the principles and techniques are equally applicable to the hardware-based acceleration of the other applications. Indeed, the hardware and software we produce can be directly incorporated into Adobe Illustrator and Adobe Premier without change. Other third party tools also use Adobe-style plug-ins and these can also immediately benefit from our hardware-based filter accelerator.

Photoshop is a widely used image processing package which provides a modular architecture for extending its functionality based on plug-ins. Images to be processed are often in true-colour (24-bits) and may be sampled from a photograph or video camera at a high resolution. Photoshop provides filters that can manipulate an image in various ways including colour manipulation and filtering (e.g. Gaussian blur). For large images these filters can take a long time to run, and there is already a market for specialised DSP-based cards which can be used with plug-ins to accelerate Photoshop. The work presented in this paper has been carried out using Adobe Photoshop version 5.0.

By using a FPGA-based co-processor system, one can produce filters that are accelerated using specialised circuits that operate at hardware speeds. One can distribute image processing circuits as plug-ins, making them a commodity item that is conveniently packaged. If high speed filters can be produced then there may be a market for FPGA-based boards in the desktop publishing niche.

Adobe Photoshop provides a collection of 'filters' that perform various image processing operations. The filter menu of Photoshop is shown in Fig. 1 below. The available filters are not fixed, but instead are read as plug-ins to Photoshop. This means that a user can purchase or develop more filters and extend the functionality of Photoshop without access to the source code of the application.

We used the publicly available Photoshop Software Development Kit (SDK) to implement a variety of filters that use the Virtex FPGA to reduce image processing time. We have been concentrating on colour space conversion (RGB to greyscale conversion) and convolution style calculations (e.g. Gaussian Blur). Gaussian Blur is one of the slowest operations in Photoshop and is often used as a benchmark when assessing the performance of desktop publishing systems.

Fig. 1. The Filter Menu of Adobe Photoshop

The filter plug-ins were developed in C++ and compiled as Windows dynamic link libraries (DLLs). The binary programming information bitstream for the accelerators is

compiled into the DLL allowing the hardware and software to be delivered in one convenient package.

3 Accelerating Photoshop Filters

Photoshop filters communicate with Adobe Photoshop using a series of messages that specify the nature of the image to be processed, as shown in Fig. 2.

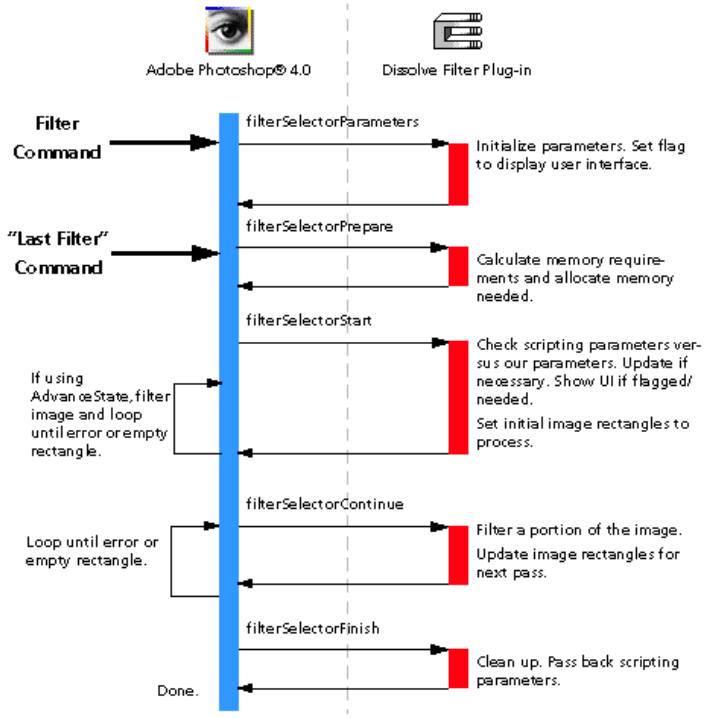

Fig. 2. Photoshop Plug-In Architecture

When Photoshop starts up it scans a series of directories containing plug-in DLLs and registers them (adding menu options to the Filter menu for each filter). Plug-ins respond to filter commands as shown in Fig. 2 which results in a series of messages begin passed, specifying the size and nature of the image to be processed.

The plug-in software that we produced asks for the image to be presented in red/green/blue/alpha format (32-bits). Every time one of our filters is selected Photoshop

copies its internal working image into a buffer for our use. We then manipulate this buffer to calculate a new image which is placed into a destination buffer. When the filter completes, Photoshop copies the destination buffer back into its own internal buffer (which requires it to reformat the image to the internal representation). Consequently for both the software and hardware filters we incur an image movement cost that is not under our control. We do not instrument the cost of copying these buffers.

The filters that dispatch the image to the FPGA co-processor just pass the image data directly to the hardware and then read back the processed results into the destination buffer.

3.1 Filters used for Performance Measurements

We decided to use two filters in our measurements: one of O(1) computational complexity and one of O(n^2). The first one is a colour to greyscale filter and the second a 5x5 convolution with loadable weights. For each filter we made a software and hardware implementation.

The greyscale filter in hardware uses 22 slices (1% of an XCV300) and can process 100 million pixels per second. The 5x5 convolver circuit takes up 2790 slices (90% utilisation) and 12 BlockRAMs (out of 16). It can process 33 million pixels per second. Both designs were specified in VHDL, synthesised without much consideration for optimisation and compiled without any layout constraints.

The convolver buffers 4 lines of image data which means that although each pixel has to be multiplied and added to 24 other pixels we need only communicate each pixel once to the FPGA. The core of the convolver then has high speed access to the required pixels held in BlockRAMs.

4 The Photoshop Co-Processor Hardware

The co-processor hardware consists of a PCI-Pamette card and a daughtercard using the Xilinx Virtex FPGA. Both cards were developed at the Systems Research Center of Compaq Computer Corp.

4.1 The PCI-Pamette

The PCI-Pamette is a generic PCI-card based on reconfigurable logic [3]. One Xilinx XC4010E FPGA implements a master and slave PCI interface supporting 32- and 64-bit transactions and contains a DMA engine capable of transferring data at full PCI-bus speed. The card features a PCI mezzanine card connector (PMC) for daughtercards.

4.2 The Virtex Daughtercard

The daughtercard is based on the new Xilinx Virtex FPGA series [8]. Fig. 3 shows a block diagram of the card and Fig. 4 a photograph. The daughtercard consists of the following components:

- 1 XCV300 Virtex FPGA in a BGA package
- 2 independent banks of synchronous ZBT SRAM (18-bits wide), 1 MB total
- 2 independent banks of synchronous DRAM (16-bits wide), 4 MB total
- programmable clock generator
- general-purpose 68-pin connector for input/output (34 signal pins)
- 64-bit PCI interface to PCI-Pamette
- clock buffer, 2.5 V switching power supply, temperature sensors

We use a Xilinx Virtex XCV300 FPGA in a Ball-Grid Array package. It is connected to a total of 5 MB of memory. We use new Zero-Bus Turnaround synchronous SRAMs [9] and more traditional synchronous DRAMs. The four independent banks of SRAM and DRAM allow for an aggregated memory bandwidth of over 1 GB/s.

The daughtercard has a flexible clocking scheme. Based on a 20 MHz oscillator or a signal from the PCI-Pamette [3] the clock generator is capable of generating any frequency between 0 and 90 MHz. This can be multiplied using the delay-locked loop circuits (DLL) of the Virtex FPGA to generate higher frequencies. The resulting signal is distributed to the RAMs and back to the FPGA itself by an external clock buffer. Using another DLL in the FPGA, we can generate a zero-skew copy of the board clock for the FPGA circuit. There are two additional clock sources available: a copy of the PCI clock, generated by a phase-locked loop on the PCI-Pamette and a clock signal coming from one of the FPGAs on the PCI-Pamette. The latter can be used, for instance, to implement a software clock.

A switching power supply is used to generate the FPGA core supply voltage of 2.5V. Two temperature sensors [2] are used to monitor the ambient temperature of the front and the back of the card. When mounted on a PCI-Pamette, most components of the daughtercard face the components on the PCI-card. The temperature sensors can be used to provide a shutdown function, should the boards get too hot.

4.3 Interface to PCI-Pamette and the Host-PC

A 64-bit PCI interface (98 wires) is provided through 3 of the 4 connectors (see Fig. 3). Nothing about the interface is PCI-specific except the pin-out as prescribed by the standard. PCI-Pamette is capable of transferring data to and from memory at a sustained 120 MB/s by using DMA transfers. This rate depends on the host-bridge and the system bus speed used in the PC. The figures we report here are for an Intel 440BX AGP chipset [5] running the system bus at 100 MHz.

The PCI-Pamette is configured with an interface, which passes data back and forth between the Virtex daughtercard and the PC's host memory. Currently, only programmed IO is supported, but a DMA mode is under development. The programmed IO mode allows for data transfer speeds of 50 MB/s to and 10 MB/s from the card. The interface is also used to connect to the configuration port of the Virtex FPGA.

4.4 Software Interface

To configure the Virtex FPGA with a circuit, we first configure the PCI-Pamette with the aforementioned interface and then download the accelerator to the Virtex FPGA. The software can move data to and from the daughtercard by writing to or reading from an address in PCI-space.

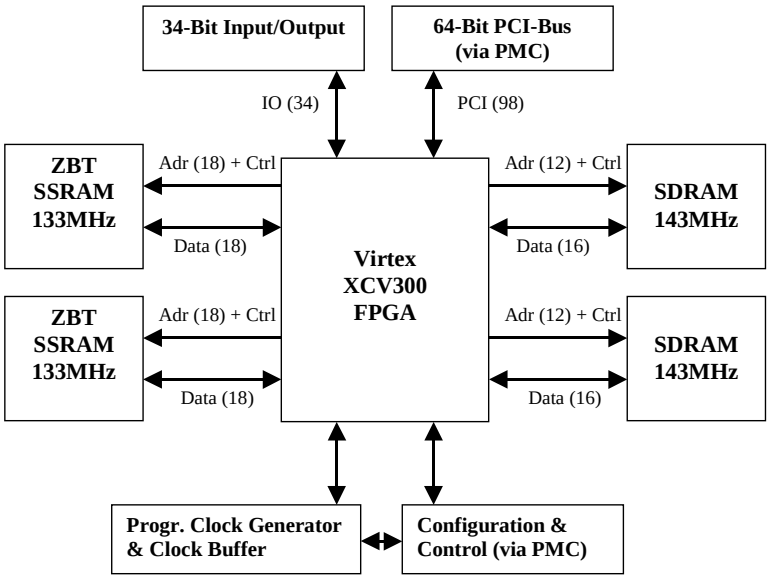

Fig. 3. The Virtex daughtercard architecture

5 Performance Analysis

5.1 Theoretical Performance

In the following, we calculate the maximal performance of a 5x5 convolution, implemented in software on a PC and as a circuit on the co-processor, respectively. For the analysis we consider a 6" x 4" photograph scanned at 600 dpi in 24-bit RGB colour (padded to 32 bits per pixel). This results in a 35 MB image of 8.6 million pixels, which has to be transferred from main memory to the computing device and back. Every colour

value of every pixel is subjected to the filter, which results in 25 multiplications and 24 additions per colour.

On the PC, transferring the image from memory to the CPU and back takes 70 MB / 800 MB/s = 88 ms, or 1.25 ns/B. If we could fit all filter coefficients into the CPU's registers (this is not possible on a Pentium-III) we could do one multiply-accumulate step every 1.25 cycles. Ignoring all other overhead, calculating the filter takes 31.25 cycles per byte. At 500 MHz this is 187.5 ns per pixel or 1.62 seconds for the whole image. Thus, the time of filtering is dominated by the calculation time. This results in a theoretical performance of 5.3 million pixels per second.

Fig. 4. A photograph of the Virtex daughtercard

On the daughtercard, transferring the image takes 70 MB / 120 MB/s = 583 ms, or 8.3 ns/B. We can fit the 5x5 filter completely in the Virtex FPGA and can therefore get 75 MACs (multiply-accumulates) per cycle. At 66 MHz this is 15.2 ns per pixel or 131 ms for the whole image. Since the performance of the hardware filter is dominated by memory transfer speed we achieve a throughput of 14.8 million pixels per second. Taking into account the transfer speed this gives us a factor of 3 improvement over the PC.

We have to emphasize that in the above calculations, very optimistic assumptions have been made for the performance of the software. In reality, we should be able to achieve higher speedups.

5.2 Measurements and Analysis

Approximate performance measurements are shown in Table 1. The measurements were

Table 1. Performance Measurements (in mega-pixels per second)

Filter	200K Pixels	3.4M Pixels
Greyscale (Software)	4.58	4.6
Greyscale (Hardware)	1.07 (speedup 0.23)	1.06 (speedup 0.23)
Convolver 5x5 (Software)	0.73	0.79
Convolver 5x5 (Hardware)	1.08 (speedup 1.48)	1.08 (speedup 1.37)

instrumented by calculating the total time taken to communicate the image data to and from the FPGA card, as well the time spent processing the image and performing the filter protocol commands. The software measurements were taken on a high-end personal computer with a 500MHz Pentium III processor and 192MB of memory. Each filter was executed five times for each image and an average of the processing time was calculated.

The measurements show that the calculations performed on the FPGA subsystem are memory bound since the 5x5 convolver [$O(n^2)$ i.e. 75 multiply-adds per pixel] proceeds roughly the same number of pixels per second as the far less computationally challenging greyscale operation [O(1)]. As expected, the software version of the convolver is compute bound, operating six times slower than the greyscale filter.

We also instrumented the built-in 5x5 convolver provided by Adobe and it performed at 1.13 mega-pixels per second. This shows that the plug-in version of the convolver is not much slower than the highly optimised built-in version. Indeed the built-in version only achieves a fifth of the theoretical throughput.

Although the results show very modest speedups, the relative results are encouraging. The memory transfer speed to the co-processor board is currently only 8 MB per second, but this can be improved by an order of magnitude using DMA. This would result in a speedup of 14 over our convolver plug-in or a speedup of 10 over the built-in version.

6 Related Work

FPGA-based Photoshop accelerators have been previously designed and built by the authors using a XC6200 [7] based system on a PCI-card [6] which contained 2MB of SRAM. This system demonstrated how Photoshop, plug-ins, device drivers, programming bit-streams and run-time control could be used to accelerate commercial applications with FPGAs. However the XC6200 system used a considerably smaller FPGA device which lacked on-chip memory blocks. Furthermore, the previous system required carefully hand-crafted filters with each cell being manually placed. Our current system is synthesised from a high level description and contains no location constraints (except for the IOs).

7 Summary and Conclusions

This project is still in progress and we have just presented preliminary results. The speedups at this stage of the project are very small, but with further enhancement of the communication infrastructure we expect speedups that make a FPGA-based Photoshop accelerator board competitive with high-end workstations or DSP-based co-processor boards. Not all types of filtering operations are suitable for acceleration with a FPGA-based co-processor and one has to carefully analyse the computational complexity of the filter, its memory requirements and its memory access behaviour.

The experiments we have performed show that the hardware solution scales linearly since the pixels per second performance for a large image is no worse than for a much smaller image. The experiments also show that the measured performance of both the hardware and the software is significantly less than the theoretical values that we have calculated. However, the hardware solution suffers from limited memory bandwidth, which we can improve in the future.

The filters that we have designed and implemented can be directly used to accelerate other Adobe Photoshop applications like Illustrator (for producing drawings) and Premier (for processing moving images). Although we have concentrated on the acceleration of a specific application using an FPGA-based co-processor we believe the general technique is applicable to a wide class of problems. Many modern computer applications are designed to use plug-ins to extend their capability after shipment, e.g. Netscape Communicator. Applications that require a significant amount of computation per datum after transfer are good candidates for acceleration using our technique. Examples include encryption/decryption, encoding/decoding and compression/decompression.

The measurements here have been performed without fully exploiting all the features of the Virtex daughtercard. For example, we could use the on-board SRAMs to cache large portions of the image to implement larger filters (e.g. to time-multiplex a 7x7 convolver which would not otherwise fit into the FPGA). Alternatively we could use the SRAMs to convolve images which are too wide to fit into BlockRAMs.

References

[1] Compaq Professional Workstation AP200 Series. Compaq Computer Corp. 1998. http://www.compaq.com/products/workstations/ap200/index.html

[2] DS1820, 1-Wire™ Digital Thermometer. Dallas Semiconductor Corp. 1998. http://www.dallassemiconductor.com/Prod_info/Thermal/thermal.html#1820

[3] PCI Development Platform. Compaq Computer Corp. 1996. http://www.research.digital.com/SRC/pamette

[4] J.D. Foley, A. Van Dam. Computer Graphics: Principles and Practice. Addison Wesley. 1997.

[5] Intel 440BX APGset. Intel Corp. 1998. http://developer.intel.com/design/chipsets/440bx/index.htm

[6] Satnam Singh and Robert Slous. *Accelerating Adobe Photoshop with Reconfigurable Logic*. FCCM'98. Napa, California. IEEE Computer Society Press, 1998.

[7] Xilinx. XC6200 FPGA Family Data Sheet. Xilinx Inc. 1995.

[8] Xilinx Virtex™ 2.5V FPGA Product Specification. Xilinx Inc. 1998. http://www.xilinx.com/products/virtex.htm

[9] Pipelined ZBT™ Synchronous Fast Static RAM. Motorola, Inc. 1998. http://motsps.com/products/memory/srams/synchronous/zbts/index.html