A Software Controlled Hardware Acceleration Architecture for Image Processing Using an Embedded Development Board

Mauricio Paz Valverde
SEED-Lab
Area of Computer Engineering &
Electronics Engineering School
Costa Rica Institute of Technology
Cartago, Costa Rica
Email: mapaz@ic-itcr.ac.cr

Jeferson González Gómez
SEED-Lab
Area of Computer Engineering &
Electronics Engineering School
Costa Rica Institute of Technology
Cartago, Costa Rica
Email: jgonzalez@itcr.ac.cr

***Abstract*—Image processing is a computer technology that has been used in various applications such as biometric authentication, surveillance and social media. It is the process of obtaining information from an image by applying a kernel and performing a convolution. This paper presents an implementation of a image processing system accelerated on a FPGA and controlled from a software interface. Morphological filtering was accelerated using a column parallelization technique to achieve maximum throughput. The system was implemented on an Altera Cyclone IV FPGA and was benchmarked against a software implementation using an Intel N2000 processor. The hardware design achieved a efficiency of 12 times compared to software implementation when processing a 256x256 grayscale image.**

***Keywords*—FPGA, hardware acceleration, digital image processing**

I. Introduction

Digital image processing has a broad range of applications intended to improve pictorial information for human perception [1]. It is important because it comprises a set of hardware, software and theoretical foundations to obtain the information of interest [2]. Due to its importance, a lot of work has been done to accelerate image processing and achieve faster results. Hardware acceleration is a process that has being widely used to accomplish this task.

Hardware accelerators are optimized functional blocks designed to offload specific tasks from general purpose CPUs. These blocks can perform faster than the analogous software running on a CPU, which usually works on a sequential instruction level. Field Programmable Gate Arrays are very suitable to host hardware accelerators. FPGAs are ideal platforms for image processing applications. They offer high levels of parallelism and improved performance over DSP and CPU-based platforms, and provide much lower power consumption than GPUs [3].

Several works related to hardware acceleration have been implemented to achieve faster digital image processing. Dharan et al. [4] proposed a hardware accelerated face detection system using skin color segmentation method on FPGA. Their algorithm was implemented using stream-oriented hardware architecture. Their results showed that a hardware design achieved up to 250 times faster than software implementation. Vourvoulakis and Kalomiros [5] developed a custom low-cost circuit board appropriate for machine vision. Their board was based on Altera's Cyclone IV FPGA and were able to applied image filters and video processing using a 640x480 resolution. Siddiqui et al. [6] proposed a high performance scalable processor for image processing, that operates at 526 MHz. The key concept of their implementation was to use multiple processors in SIMD mode to achieve data level parallelism. The system was reported to perform up to 33 times faster, implementing the Traffic Sign Recognition Algorithm.

In this paper we present an implementation of image processing algorithms hosted on a Terasic DE2i-150 development board. The objective is to propose a filter hardware accelerator that can be controlled through a software interface. The proposed system was made possible due to the Terasic board design, since it has an embedded CPU and a FPGA, both communicated by a PCIe channel. The platform architecture allows implementing high performance tasks in hardware, and to develop software interfaces running in the CPU to control the hardware. Therefore it is possible to configure and handle the hardware acceleration process during software execution time, providing a lot of flexibility for the applied design.

To gauge the performance of the hardware system in terms of execution speed, the image processing was also implemented in full software. This execution was measured to obtain low level performance metrics, specifically virtual clock cycles, to compare against the hardware implementation performance.

This paper is sectioned as follows. Section II details the theoretical foundations of the implemented filters. Section III covers the system overview. Section IV describes the implementation of the hardware acceleration process. In section V the designed benchmark is described, followed by section

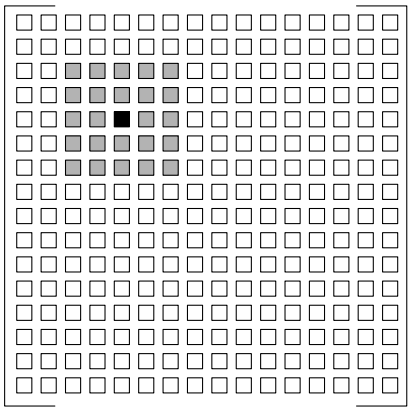

Fig. 1. 5x5 Mask on a 16x16 pixel image.

VI that presents the experimental results. Finally section VII concludes the paper.

II. Minimum and Maximum Filters

Minimum and maximum filters, also known as erosion and dilation, are morphological filters that work by taking into account a neighborhood around each pixel. A dilation filter outputs the maximum pixel value in the corresponding neighborhood, whereas an erosion filter returns the minimal value [7]. In other words, each pixel within the kernel will be compare against the others to decide which has the biggest or the smallest value.

Morphological techniques probe a small shaped image or template called a structuring element or kernel. The kernel element is positioned at all possible locations in the image to extract the corresponding neighborhood of pixels [1]. For the current implementation the kernel is a 5x5 rectangular shape; therefore all twenty five pixels in the neighborhood are going to be considered to calculate the new pixel value.

In Fig. 1 is an example of a 5x5 kernel in a 16x16 small image. The pixel with the maximum or minimum value, depending on the applied filter, will replace the anchor pixel, in this case the black pixel in the kernel center.

Pixels on the image borders will be ignored because if the kernel anchor is on a border pixel, part of the kernel will be outside the original image. For this reason, these pixels will keep their original values and are not going to be processed as kernel centers, causing a reduce in algorithm complexity.

III. System Design

Diagram in Fig. 2 shows the high level proposed system. The whole system will run on a Terasic DE2i-150 development board, which is an embedded platform that combines an Intel CPU coupled with an Altera Cyclone FPGA via a dual PCIe channel. The board design allows to offload custom intensive operations to the hardware for process acceleration.

On the CPU side a GNU/Linux operating system provides the software interface for the user to program the hardware

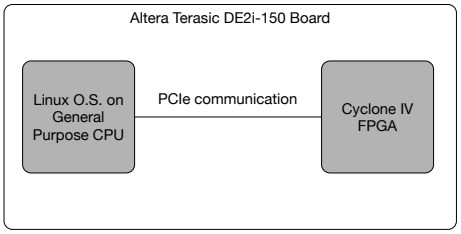

Fig. 2. High Level System Structure

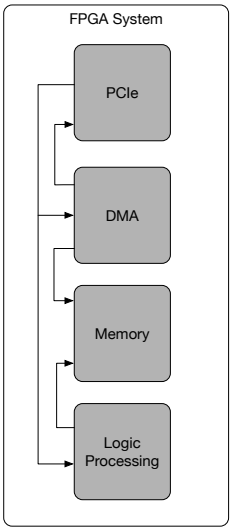

Fig. 3. Hardware Architecture

and execute the filters. The O.S. is specifically CentOS 6.7, the reason for choosing this O.S. is to take advantage of the PCIe kernel driver that is available for this specific distribution.

A. Software Interface

In order to communicate with the FPGA a software interface must exist to transmit and receive data through the PCIe channel. Driver communication methods are accessed by means of a designed C/C++ program. The main purpose of the software interface is to allow users to choose an image, select a filter and display the new calculated image. The software interface is responsible of extract the selected image pixels, initiates the transfer, and then retrieves the generated data by the hardware filter.

B. Hardware Architecture

A high level architecture diagram of the hardware implementation is illustrated in Fig. 3. The system is mainly integrated by three Altera IP cores and a custom logic design. As the image shows there are four main blocks that comprise the hardware system. The PCIe block sends and receives data through the board channel and establishes the communication link with the CPU endpoint. A Scatter-Gatter DMA is used to

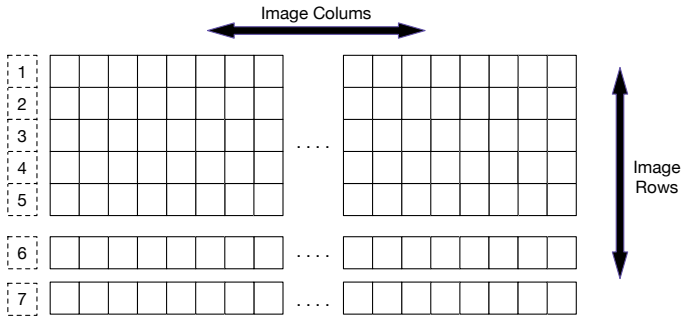

Fig. 4. Register banks

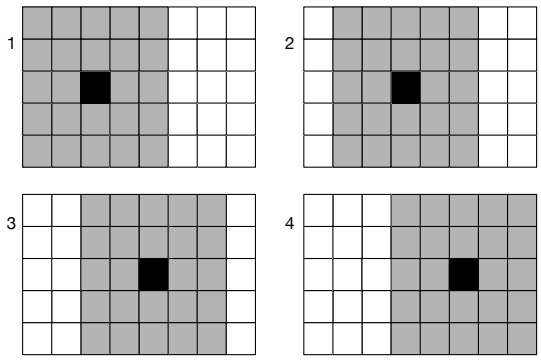

Fig. 5. Kernel shift through register bank

efficiently move data between the PCIe module and memory, and finally the logic processing block reads data from memory to process the pixels and generated a filter image, which will be send to the CPU in reverse order. The modules interconnection was handled by the Qsys tool included in the Quartus Prime Altera software [8].

IV. Implementation

A. Memory

The system memory is a 128kb on-chip dual port RAM distributed in 16k words by 64-bit wide each. A memory word can store up to eight pixels. Due to memory specification, the maximum allowed image size for processing is 64kb, which is about a 256x256 grayscale image. The process was designed to use half of the memory to store the original image and the other half to store the processed data.

Bus arbitration between different modules that need access to memory such as DMA and custom logic were handled automatically by the Qsys interconnection, hence there was no need to develop a bus controller to restrict same time access to memory.

B. Hardware Filters

In order to accelerate the image processing and reach a faster execution time, the system requires enough available data from memory at run time to parallelize the filter logic process. To achieve this, register banks were used to store pixels content.

Each register bank was configured to store eight pixels of five different image rows. A series of cascaded banks were instantiated to have five complete image rows ready to process in each clock cycle. The require amount of register banks instances depends on the image size. For a specific image, the system needs $\frac{imgColumns}{8}$ register banks. Figure 4 illustrates the system bank configuration. The numbers on the left represent row numbers.

When the image processing starts, the first two register banks begin loading data from memory. Once the load is completed, the memory bus can be released to allow the next

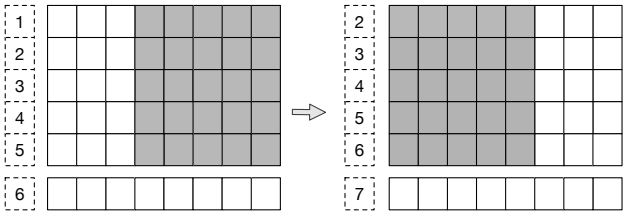

Fig. 6. Kernel shift through image rows

register banks to retrieve their data. This process continues until the first five image rows are stored.

Filter logic starts to be applied once enough retrieved data is available. The filtering process is illustrated in Fig. 5, the anchor kernel shifts to the right of the register bank to cover four 5x5 neighborhoods of stored pixels. As the image shows, four values can be calculated from a single register bank using a 5x5 kernel.

Once the filtering process has been applied to all register banks, the next image row has to be read. This new data will shift up the first row of each bank, since these pixels are no longer needed. In this way, no more hardware is needed to process the image. The process is illustrated in Fig. 6, row one values were overwritten by the new values of row six, then row two values will be replaced by row seven values, and so on. Additionally, for each load the kernel is shift through the updated neighborhood of pixels to calculate new pixel values.

V. Benchmark

In pursuance of measuring our hardware's performance, a benchmark was designed to compare the hardware filters discuss in this paper against a software equivalent implementation. A program on CentOS ran the same filters that were implemented on our hardware, and accessed specific CPU registers to obtain performance data. OpenCV and PAPI libraries were used to apply the image filters and get the performance data, respectively.

```
1 kernel = getStructuringElement(
2                             CV_SHAPE_RECT,
3                             Size(kernel_length,
4                             kernel_length));
5
6 erode(src_img, erode_img, kernel);
7
8 dilate(src_img, dilate_img, kernel);
```

Listing 1. OpenCV Erode and Dilate Methods

A. OpenCV Implementation

OpenCV is a library that includes hundreds of computer vision algorithms, it is essentially a C++ API that has a modular structure, which means that each package includes several libraries [10].

Listing 1 shows a C++ code snippet of the erosion and dilation OpenCV filters, equivalents of the hardware implementation. Both methods require three parameters: the source and destination images and the structuring element(kernel).

B. P.A.P.I. Library

The Performance Application Programming Interface provides access to hardware counters found on most modern microprocessors. It can be used to collect low level metrics such as clock cycles, instruction counts and many others [9].

During software filter execution, virtual clock cycles were measured. This clock counter increases only when the program is executing in user space mode. Therefore, it only includes cycles related to the filtering process.

The low level function used to retrieve the virtual clock cycles was $PAPI\_get\_virt\_cyc()$. The function measures from an arbitrary starting point. For example, to calculate the virtual clock cycles of the erosion filter shown in listing 1, the reference was set on line 5 and the result was obtained on line 7.

VI. Analysis and Results

The system software components were implemented on a Linux environment, using C/C++ language. Whereas the hardware component were designed using Verilog HDL in Altera Quartus Prime software. The resulting image for the dilation filter is depicted in Fig. 7, the image is brightened by the filter, as expected. The differences between the resulting images obtained from hardware and software approaches were zero.

For the benchmark implementation, a C++ program was developed to run on the embedded CPU board. The CPU was an Intel Atom Dual Core Processor that runs up to 1.6 GHz. A script was designed to run ten thousand executions of the same program and to collect the require information using PAPI library. A statistical analysis of the timing results was develop to obtain precise data. In hardware, a register was configured to count clock cycles during filtering process and once the process finishes, the system displays the result in the board's 7-segment displays. Experimental results are presented in Table. I. The hardware solution took a lot less clock cycles to apply the image filtering than the software solution, the speed-up achieved was around 11 times, as illustrated in Eq. 1.

TABLE I
Original Execution Results

	Total Cycles	Total Time
Software Filter	2281403	1.424 ms
Hardware Filter	197361	3.947 ms

TABLE II
Possible Execution Results

	Total Cycles	Total Time
Software Filter	2281403	1.424 ms
Hardware Filter	197361	1.315 ms

Fig. 7. Dilation Filter Results

$$\frac{2281403}{197361} = 11.56 \tag{1}$$

Another interesting fact about the obtained results, are the execution times. Despite that the hardware implementation took less cycles to process the image, it has a bigger execution time. This is because the Intel CPU has a much faster clock frequency, hence it process more data in less time than the FPGA. However, this is the reason because time comparison is not a reliable metric. After all, if the board oscillator would have a higher frequency the execution time would be smaller. According to the Quartus report, the maximum frequency this specific hardware can handle is up 150 MHz, this means a frequency 3 times faster than the actual clock frequency, 50 Mhz. Table. II presents the results if the hardware system would be running at clock cycle of 6.6ns, corresponding to 150 MHz. With this improvement in clock frequency the hardware system is better in both total cycles and total time, as expected.

VII. Conclusion

In this paper, a software controlled hardware architecture for image processing has been presented. The hardware implementation was successfully controlled and configured from the software interface. The design has been fully implemented and tested on a Terasic DE2i-150 board, with the implementation of the maximum and minimum filter algorithms. Given the results of the acceleration, the hardware was more efficient than

the software reference, by roughly 12 times. The performance was compared to an equivalent system implemented in full software using OpenCV filter methods.

Hardware original timing results can be improved by a clock frequency enhancement. Using a faster clock can level up both implementations. Another possible way would be to improve the hardware design to decrease time delays and be able to reach faster clock frequencies.

REFERENCES

[1] D. G. Bailey, *Design for Embedded Image Processing on FPGAs*. Wiley-IEEE Press, 2011.

[2] I. N. Rodrigues, C. L. S. de Melo, V. Mello da Frota Botinelly, and J. P. de Oliveira, "FPGA hardware architecture with parallel data processing to detect moving objects using the background image subtraction technique," in *2015 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*. IEEE, oct 2015, pp. 841–846. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7404670

[3] S. Kestur, J. D. Davis, and O. Williams, "BLAS Comparison on FPGA, CPU and GPU," in *2010 IEEE Computer Society Annual Symposium on VLSI*. IEEE, jul 2010, pp. 288–293. [Online]. Available: http://dl.acm.org/citation.cfm?id=1848074.1848496

[4] S. V. Dharan, M. Khalil-Hani, and N. Shaikh-Husin, "Hardware acceleration of a face detection system on FPGA," in *2015 IEEE Student Conference on Research and Development (SCOReD)*. IEEE, dec 2015, pp. 283–288. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7449341

[5] J. V. Vourvoulakis, J. Lygouras, and J. A. Kalomiros, "Acceleration of Image Processing Algorithms Using Minimal Resources of Custom Reconfigurable Hardware," in *2012 16th Panhellenic Conference on Informatics*. IEEE, oct 2012, pp. 68–73. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6377369

[6] F. M. Siddiqui, M. Russell, B. Bardak, R. Woods, and K. Rafferty, "IPPro: FPGA based image processing processor," in *2014 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, oct 2014, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6986057

[7] J. Stein, *Digital Signal Processing: A Computer Science Perspective*. Wiley-Interscience, 2000.

[8] *Quartus Prime Standard Edition Handbook*, Altera Corporation, May 2016. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/qts/qts-qps-handbook.pdf

[9] (2016, May) PAPI library overview. [Online]. Available: http://icl.cs.utk.edu/projects/papi/wiki/PAPIC:Overview

[10] (2016, May) OpenCV documentation page. [Online]. Available: http://docs.opencv.org/3.1.0/