International Journal of Technology 10(7): 1315-1325
ISSN 2086-9614

HARDWARE-BASED SOBEL GRADIENT COMPUTATIONS FOR SHARPNESS ENHANCEMENT

Daniel C.K. Kho[1], Mohammad Faizal Ahmad Fauzi[1], Sin Liang Lim[1*]

[1]*Faculty of Engineering, Multimedia University, Persiaran Multimedia, 63100 Cyberjaya, Selangor, Malaysia*

(Received: November 2018 / Revised: January 2019 / Accepted: August 2019)

ABSTRACT

The majority of imaging systems are software based; they require some kind of microprocessor or microcontroller for the imaging algorithms to run. As the speed requirements of imaging and communications systems increase, the need for more hardware-based imaging systems arises. These fully hardware systems solve the fundamental problem inherent in software-based solutions, in which the speed of the algorithms depend on the instruction cycle speed of the processor. Once an algorithm is designed directly on hardware, the speed of the algorithm depends on the system clock frequency and the propagation delays of the logic cells (or standard cells) used in the design, usually measured in nanoseconds per cell. Therefore, such systems no longer depend on any instruction cycle delays, as there is no microprocessor involved. Most modern imaging and communications systems rely on digital signal processing (DSP) to compute complex mathematical operations. The emergence of powerful and low-cost field-programmable gate array (FPGA) devices with hundreds of arithmetic multipliers has enabled the development of many such DSP hardware applications, traditionally implemented only as software solutions.

Keywords: Digital signal processing; Edge detection; Gradient; Sobel; VHDL

1. INTRODUCTION

Lately, there have been several texts (Li & Chu, 1997; Nelson, 2000; Yasri et al., 2009; Mehra & Verma, 2012; Nosrat & Kavian, 2012; Sanduja & Patial, 2012; Singh et al., 2012; Umar et al., 2012; Bhagat et al., 2015) written on hardware-based Sobel implementations on FPGAs using VHDL (Ashenden, 2008) or Verilog. However, nearly all of these advocate the use of calculating the gradient magnitude by obtaining the sum of the absolute values of the gradient in both the horizontal and vertical directions. Implementing gross approximations of many such nonlinear imaging algorithms (Arce et al., 2000; Aubert & Kornprobst, 2006; Bertalmıo et al., 2001; Chambolle, 1994; Kokkinos, 2013; Kornprobst et al., 1999; Mitra & Sicuranza, 2001; Xu & Mueller, 2010) on hardware has become common practice.Although this approach simplifies the hardware implementation by avoiding the more computationally intensive square root calculations, the resulting gradient magnitude suffers from having more errors than a gradient magnitude calculated using the Pythagorean theorem of square-rooting the sum of squares of the gradients in each horizontal and vertical direction.

Before other algorithms are performed, usually, an image filter is applied. This preprocessing filter helps ease the computation of further downstream algorithms, such as those used in optical

[*]Corresponding author's email: lim.sin.liang@mmu.edu.my, Tel. +60-383125366, Fax. +60-383183029
Permalink/DOI: https://dx.doi.org/10.14716/ijtech.v10i7.3263

character recognition systems (Pangestu et al., 2017), or the K-NN algorithms (Naik & Metkewar, 2015) used in artificial intelligence. Either a spatial filter, such as a Sobel edge detector, or a histogram equalizer frequency domain filter may be used as the prefilter, depending on the type of further processing required.

This paper introduces a computationally efficient technique of preserving the precision of the gradient magnitude by using an efficient and fast square root algorithm in the computation of the gradient magnitude. Although we also introduce a different kernel processing scheme that computes kernels in parallel, this paper focuses its discussion on the use of the fast reciprocal square root (FRSR) algorithm for hardware-based Sobel edge detection.

2. METHODS

2.1. Background Theory

In an *n*-dimensional rectangular coordinate system (Kreyszig, 2011), the gradient of a scalar function $f(x_1, x_2, \ldots, x_n)$ is the vector field $\mathbf{f}(x_i\,|_{i\in(1,n)})$ whose components are the partial derivatives of *f*:

$$\begin{aligned}\nabla f &= \frac{\partial f}{\partial x_1}\widehat{\mathbf{e}}_1 + \frac{\partial f}{\partial x_2}\widehat{\mathbf{e}}_2 + \ldots + \frac{\partial f}{\partial x_n}\widehat{\mathbf{e}}_n \\ &= \sum_{\forall n>0\in\aleph}\left(\frac{\partial f}{\partial x_n}\widehat{\mathbf{e}}_n\right)\end{aligned}, \tag{1}$$

where $\hat{\mathbf{e}}_i$ represents the orthogonal unit vectors pointing in the coordinate system's directions. Equation 1 is the general equation of the gradient of a scalar function in an *n*-dimensional rectangular coordinate system.

For a 3D Cartesian coordinate system, as in the case of our physical world, Equation 1 becomes

$$\begin{aligned}\nabla f &= \frac{\partial f}{\partial x}\hat{\mathbf{i}} + \frac{\partial f}{\partial y}\hat{\mathbf{j}} + \ldots + \frac{\partial f}{\partial z}\hat{\mathbf{k}} = \left(\frac{\partial}{\partial x}\hat{\mathbf{i}} + \frac{\partial}{\partial y}\hat{\mathbf{j}} + \ldots + \frac{\partial}{\partial z}\hat{\mathbf{k}}\right) f \\ &= \begin{bmatrix}\frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial z}\end{bmatrix}\begin{bmatrix}\hat{\mathbf{i}} \\ \hat{\mathbf{j}} \\ \hat{\mathbf{k}}\end{bmatrix} = \begin{bmatrix}\frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z}\end{bmatrix}\begin{bmatrix}\hat{\mathbf{i}} \\ \hat{\mathbf{j}} \\ \hat{\mathbf{k}}\end{bmatrix} f\end{aligned}, \tag{2}$$

where $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, and $\hat{\mathbf{k}}$ are the orthogonal unit vectors in the *x*, *y*, and *z* axes, respectively. For simplicity, some texts denote the partial derivatives pointing in each of the coordinate system's directions as

$$g_x = \frac{\partial f}{\partial x}; \quad g_y = \frac{\partial f}{\partial y}; \quad g_z = \frac{\partial f}{\partial z}. \tag{3}$$

We can also express this gradient vector in the 3D spherical coordinate system in terms of its magnitude *r* and phases (azimuth φ and altitude/elevation θ). Recall that to convert from the Cartesian coordinate system (x, y, z) to spherical coordinates (r, θ, φ), the following equations can be used:

$$\begin{aligned}&r = \sqrt{x^2 + y^2 + z^2} \\ &\theta = \arccos\left(\frac{z}{r}\right); \qquad \varphi = \arctan\left(\frac{y}{x}\right)\end{aligned}. \tag{4}$$

The spherical gradient can therefore be written as

$$\begin{aligned}\nabla f &= |\nabla f|\angle(\theta,\varphi) = \mathbf{f}(x,y,z)\\ &= \sqrt{\left(\frac{\partial f}{\partial x}\right)^2+\left(\frac{\partial f}{\partial y}\right)^2+\left(\frac{\partial f}{\partial z}\right)^2}\angle(\theta,\varphi),\\ &= \sqrt{g_x{}^2+g_y{}^2+g_z{}^2}\angle(\theta,\varphi)\end{aligned} \tag{5}$$

where

$$\begin{aligned}&|\nabla f| = \sqrt{g_x{}^2+g_y{}^2+g_z{}^2}\\ &\theta = \arccos\left(\frac{g_z}{|\nabla f|}\right); \qquad \varphi = \arctan\left(\frac{g_y}{g_x}\right) = \arctan\left(\frac{\partial x}{\partial y}\right)\end{aligned} \tag{6}$$

The resulting gradient is a vector field, denoted as $\mathbf{f}(x,\ y,\ z)$ in Equation 5. The magnitude of the gradient is denoted as $|\nabla f| = |\mathbf{f}(x,\ y,\ z)|$. The gradient equations are clearly nonlinear because the square root, arctangent, and arccosine functions are all nonlinear.

However, to simplify the scope of this study, we will be working with 2D images. For a 2D Cartesian coordinate system, the gradient of a scalar image function $f(x,\ y)$ can therefore be expressed in polar coordinates, as shown in Equation 7.

$$\begin{aligned}|\nabla f| &= |\nabla f|\angle\varphi = \mathbf{f}(x,y)\\ &= \sqrt{g_x{}^2+g_y{}^2}\angle\arctan\left(\frac{g_y}{g_x}\right)\end{aligned} \tag{7}$$

In our work, we will show that calculating the gradient magnitude as the square root of the sum of squares of the gradients in the horizontal and vertical directions, $|\nabla f| = \sqrt{g_x{}^2+g_y{}^2}$, gives us a more accurate representation of the gradient information within an image, as opposed to making gross approximations of the gradient magnitude by omitting the square root computations. We will show that implementing a fast, area-efficient, and computationally efficient square root algorithm on hardware is not just feasible but also necessary for emerging imaging applications that have more stringent demands on image quality.

In the case of the Sobel operator, the partial derivatives g_x and g_y can been approximated with 3×3 kernel matrices:

$$g_x = \begin{bmatrix}-1 & 0 & +1\\ -2 & 0 & +2\\ -1 & 0 & +1\end{bmatrix} * \mathbf{F}; \qquad g_y = \begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ +1 & +2 & +1\end{bmatrix} * \mathbf{F} \tag{8}$$

In the many cases in which the absolute values of the derivatives are used to calculate the magnitude of the gradient vector, the gross approximation is applied as such:

$$\begin{aligned}|\nabla f| &\approx |g_x| + |g_y|\\ &\approx \ |(z_7+2z_8+z_9)-(z_1+2z_2+z_3)|+,\\ &\quad\ |(z_3+2z_6+z_9)-(z_1+2z_4+z_7)|\end{aligned} \tag{9}$$

where z_1 through z_9 are pixels from a 3×3 section of image data.

From the basic definition of the derivatives of a 1D function f(n), we know that the first-order derivative is the difference between two neighboring points on the same function with respect to n:

$$\frac{\partial f}{\partial n} = g_n = f(n+1) - f(n) \tag{10}$$

For our purposes, we need only to calculate the magnitude of the gradient for now. To compute the magnitude of the gradient numerically, we substitute Equation 10 into Equation 7:

$$\begin{aligned}|\nabla f| &= \sqrt{{g_x}^2 + {g_y}^2} \\ &= \sqrt{[f(x+1) - f(x)]^2 + [f(y+1) - f(y)]^2}\end{aligned} \tag{11}$$

Here, we compute the difference in intensity of adjacent pixels both in the x and y directions as the first-order partial derivatives. For our case, the intensity difference of adjacent pixels, $f(n+1) - f(n)$, may also use the definition taken from Sobel approximation, as shown in Equation 9, i.e., $g_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$ and $g_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$

Substituting this into Equation 11, we have

$$\begin{aligned}|\nabla f| &= \sqrt{{g_x}^2 + {g_y}^2} \\ &= \sqrt{[(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)]^2 + [(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)]^2}\end{aligned} \tag{12}$$

There are existing square root algorithm implementations in hardware, but they have not been applied to the field of image processing. We have studied two of these implementations—the non-restoring square root algorithm (Nanhe et al., 2013) and the FRSR (rsqrt) algorithm Lomont, 2003; Robertson, 2012; Zafar & Adapa, 2014; Kho et al., 2018). Because of the speed and efficiency of the FRSR algorithm, we have decided to use this in the design and implementation of the Sobel gradient computations.

2.2. Algorithm Modeling

The reciprocal square root (rsqrt) algorithm approximates the reciprocal of the square root $1/\sqrt{x}$ of a given number *x*. Because

$$\frac{1}{\sqrt{x}} = \frac{\sqrt{x}}{x}, \tag{13}$$

we can also obtain the square root by multiplying the reciprocal square root result with the given input.

All our equations have been modelled directly using Python. These algorithms were compared against Python's built-in OpenCV libraries. First, Equation 8 was modelled in Python to obtain the horizontal and vertical gradients, g_x and g_y respectively, of the Lenna image. Second, Equations 9 and 12 were then modelled to obtain the approximate and actual magnitudes of the gradient vector $|\nabla f|$.

The results from these Python models are dumped to files that are later compared with the results from the hardware simulations.

2.3. Hardware Architecture

2.3.1. System block diagram

Figure 1 shows the block diagram of our proposed hardware architecture for our video processing platform. For the purpose of having a hardware platform capable of showing video processing features, the first phase of our video platform involved efforts to design and build the processing and to display the blocks necessary to enable such a demonstration.

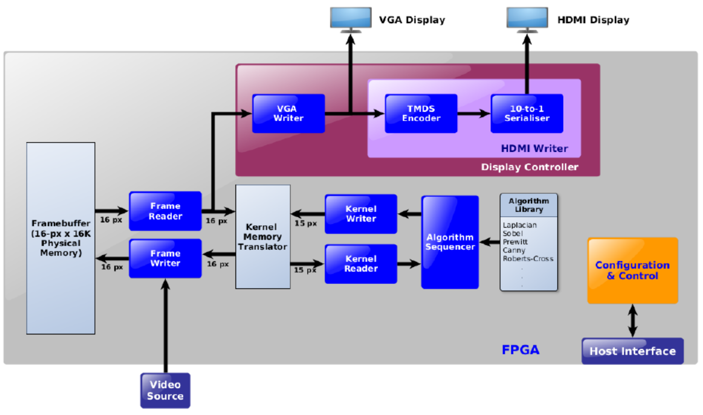


Figure 1 Block diagram of our proposed video processing hardware platform

Therefore, for the purpose of this study, we have simplified our architecture to read a fixed 512×512 pixel Lenna image stored in an FPGA's internal random access memory (RAM) blocks, instead of having a variable video source coming from a camera input. Working with static images helps us verify that our algorithms work as intended and also helps us measure the performance of our algorithms against that of other techniques. A camera input is planned for the next phase of this project and is not within the scope of the present study.

2.3.2. Frame buffer, frame reader, and frame writer

The frame buffer stores all the pixels from a single 512×512 pixel image or a 512×512 pixel segment of a larger image that needs to be processed. In our implementation, we have used the standard 512×512 pixel Lenna image to be able to have a fixed static image for analysis and comparison of our algorithms against other techniques. This frame buffer is implemented as block RAM in hardware. Because of the structural constraints of the physical memory within the FPGA, the frame reader can only read image data from the frame buffer in 2n-pixel blocks; in our case, the memory data width is 16 pixels wide. For simplicity, each pixel is 8 bits in the case of monochrome processing. In the future, our design can still be extended for color processing, in which each of the three color components will be processed independently and in parallel, in effect making our existing design a color channel processor.

2.3.3. Kernel memory translator, reader, and writer

Figure 2 shows the kernel chunk processing scheme as implemented by our current hardware architecture.

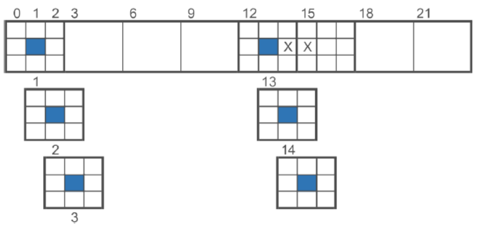


Figure 2 Kernel chunk processing scheme

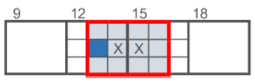


Figure 3 Boundary kernel structure

The kernel reader reads 3×3 (or odd number-sized) kernels from the kernel memory translator in three read requests. Each read request gives us 15 pixels, so we will receive 45 pixels in three such requests. This 45-pixel chunk will then be processed by the algorithm processing block in 3×3 pixel kernels before proceeding to the next 45-pixel chunk.

Each 45-pixel chunk contains full data for twelve 3×3 pixel kernels, as well as partial data for another two 3×3 pixel kernels at the boundary of the next 45-pixel chunk. To ease our processing, we have grouped these two boundary kernels, numbered kernels 13 and 14 in Figure 2, as a single 4×3-pixel boundary kernel. The data structure of our boundary kernel is shown in Figure 3. Because we shift by one pixel for every computation, we have a total of 15 full or partial 3×3 pixel kernels that can be processed from the data given by this 45-pixel chunk. In our current design, all the 15 kernels are processed in parallel.

2.3.4. Sobel using the fast reciprocal square root algorithm

Most video and image processing Sobel edge detection algorithms (Nelson, 2000; Yasri et al., 2009; Mehra & Verma, 2012; Nosrat & Kavian, 2012; Singh et al., 2012; Umar et al., 2012; Bhagat et al., 2015) use the sum of the absolute values of the gradient in both the horizontal and vertical directions, $|\nabla f| \approx |g_x| + |g_y|$, in the calculation of the gradient magnitude. However, here, we are using the square root algorithm in the gradient magnitude computations.

Furthermore, in most hardware-based square root computational systems (Li & Chu, 1997; Ercegovac et al., 2000; Ercegovac et al., 2005; Wang, 2007; Lachowicz, 2008; Sajid et al., 2010; Istoan & Pasca, 2015; Ananthalakshmi & Sudha, 2017), the non-restoring square root algorithm, or the sum of the absolute values of the gradients in the horizontal and vertical directions, is used to approximate the magnitude of the gradient vector. However, in this work, we are using the FRSR algorithm (Lomont, 2003; Robertson, 2012; Zafar & Adapa, 2014; Kho et al., 2018) to compute the square root in hardware.

3. RESULTS AND DISCUSSION

3.1. Model Verification

While working on our algorithm, we found it necessary to compare the results of our technique with those of other algorithms. However, we noticed that we could not assume there is an algorithm that gives better results compared with our algorithm. The goal of verifying our model against other algorithms is to determine how much sharper or blurrier our algorithm is against other algorithms. Because of a lack of a golden reference image that is universally accepted as the sharpest version of the standard Lenna image, we decided that we needed to perform image quality measurements without having a reference image. There are several texts (Kanjar & Masilamani, 2013a; Kanjar & Masilamani, 2013b; Kanjar & Masilamani, 2017) that discuss techniques to perform sharpness measurements, and these kinds of measurements remain an actively researched topic today.

In the case of comparing between two Sobel edge detection algorithms, the gradient magnitude masks of the resulting images from both algorithms are measured. Because Sobel algorithms are not prone to noise and false positives, the number of edge pixels detected by an algorithm determines the sharpness of the edges. False positives and the *mistakes* caused by the algorithm in the detection of edges are uncommon for Sobel algorithms. However, if an edge is represented by more pixels, the thickness of the edge increases, and the edge appears blurrier than another edge whose thickness is small. One can safely assume, having the gradient magnitudes of two competing output images from two different Sobel algorithms, that the output image with thinner edges is sharper than that whose edges are thicker.

In this study, for the sake of simplicity, we propose a fairly simple technique to measure the sharpness quality of our algorithm against another algorithm. To have a fairly good assessment of how our algorithm performs, we calculate the relative difference in the number of edge pixels from our output image against that from the output image of another competing algorithm. From the Sobel gradient output of one algorithm, the sum of the number of edge pixels, hereinafter referred to as the sum of edges, is measured and compared against the sum of edges of a competing Sobel algorithm. If the sum of edges from one algorithm is smaller than that from a competing algorithm, it is implied that the edges from the former algorithm are thinner than those from the latter. There are, of course, more complicated methods to measure the thickness of edges within a Sobel gradient mask; however, the sum of edge pixels method is a fairly inexpensive technique to estimate the performance between two edge detection algorithms.

To determine whether a pixel is an edge pixel or not, we apply a threshold to the gradient mask. In this study, we used a threshold value of 10, which means pixels having values between 0 and 10 are considered edge pixels, assuming that the gradient mask plots edges using brighter values on a dark background. From our measurements, we found that our algorithm is 0.438% sharper than the OpenCV implementation and 6.508% sharper than an algorithm that uses absolute values rather than computes the square root.

3.2. Functional Simulations and Hardware Synthesis

We have implemented this design on a Xilinx Zynq (ZC7010) FPGA device and used Xilinx's ChipScope integrated logic analyzer to acquire real-time waveforms from our development hardware. Figure 4 shows a partial view of the functional simulation results in ModelSim.

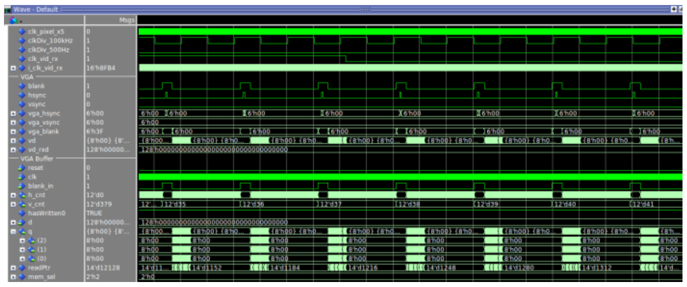


Figure 4 ModelSim simulation results

3.3. Place-and-Route (PAR), and Design Assembly

We have performed manual placement-and-routing for some speed-critical blocks, especially those that need to be connected to the Transition-Minimized Differential Signaling (TMDS) I/O pins for High-Definition Multimedia Interface (HDMI) transmission to an external display. This is shown in Figure 8.

Xilinx's tools show that our Sobel gradient algorithm was physically implemented with only 2,577 look-up tables (LUTs). This includes the FRSR algorithm as a part of the gradient calculations, as well as the kernel buffering and processing algorithms. As we are processing 15 kernels in parallel, our implementation may utilize more resources than other implementations. However, more processing is completed within a shorter amount of time. Furthermore, in our implementation, we have included other blocks, such as the frame buffer, frame reader, and frame writer, which are not included in other implementations. Table 1 shows our implementation results compared with the existing literature.

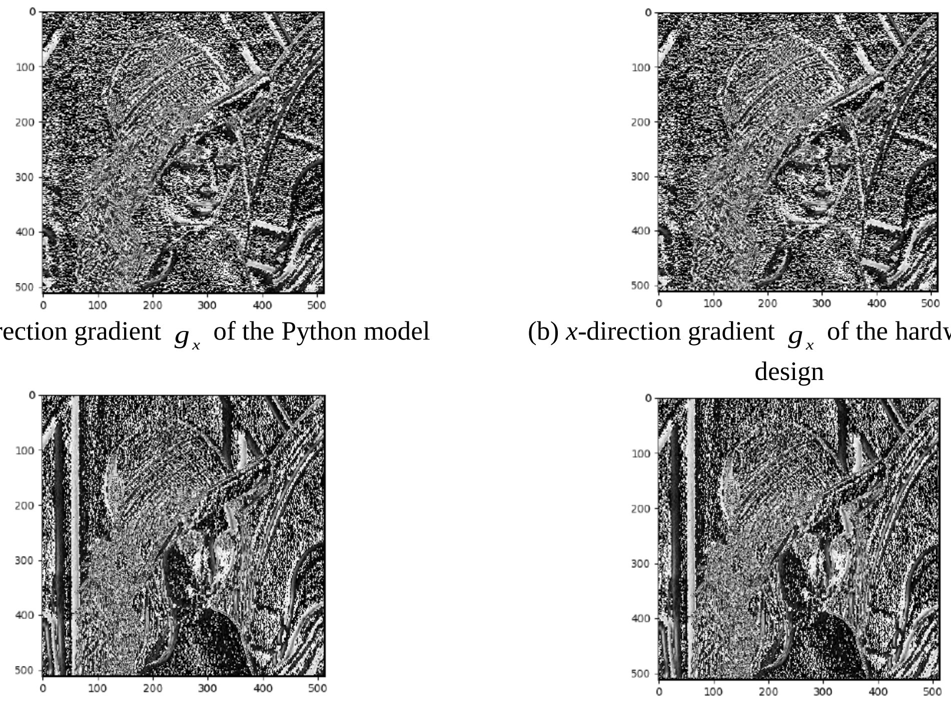
3.4. Discussion

Our Sobel gradient algorithm was verified by writing the equations directly in Python and verifying the algorithm against a built-in Python library OpenCV. The FRSR algorithm was not modelled, as the hardware functional simulations and verification processes already verify the correctness of this algorithm against the square root function.

Table 1 Xilinx post-PAR utilization report of the Sobel algorithm with fast reciprocal square root

Resource	Proposed	Sanduja	Mehra
Total LUTs	2577	3901	353
Total LUTRAM	535	N/A	N/A
Total flip-flops	3084	836	482
Total BRAM	15	N/A	N/A
DSP	10	N/A	N/A

All hardware blocks have been written in VHDL hardware language. The design was functionally simulated in Mentor Graphics ModelSim, and the data from our simulations were compared against those from the Python models. Our simulation test bench dumps files in the same format as the Python scripts to ease our correlation and debug work, as shown in Figure 5.



(a) *x*-direction gradient g_x of the Python model

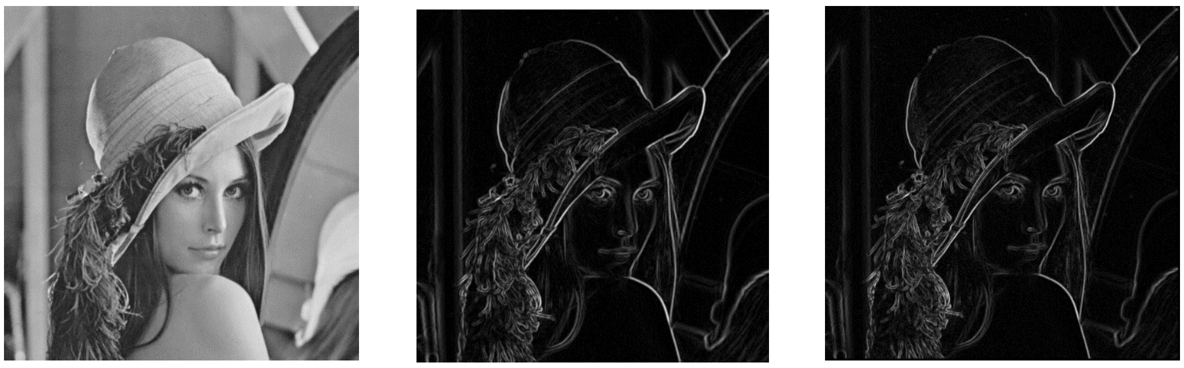
(b) *x*-direction gradient g_x of the hardware RTL design

(c) *y*-direction gradient g_y of the Python model

(d) *y*-direction gradient g_y of the hardware RTL design

Figure 5 Comparison of the *x*- and *y*-direction gradients of the hardware RTL design against the OpenCV Python model

After gaining enough confidence from our simulations, we synthesized our design to FPGA hardware using Xilinx Vivado. The final implementation was downloaded into the Xilinx Zynq 7C7010 device, and the hardware results were measured and verified. We used Xilinx ChipScope to acquire and display the data in real time.



(a) Original Lenna image. (b) OpenCV Sobel implementation. (c) Our Sobel implementation modeled and implemented on hardware

Figure 6 Comparison of the gradient processing of the standard Lenna image between our implementation and OpenCV

3.5. Future Work

Future work may explore the gradient computations of images that contain depth information, such as those from a time-of-flight camera or a 3D image, which will be making use of Equation 5. We could also explore various video input mechanisms, such as those from a camera sensor, VGA or HDMI cable input, or wireless transfer.

Future work may also discuss our kernel processing algorithms in more detail, as well as perform performance analysis and comparisons against other solutions.

4. CONCLUSION

The Sobel algorithm, used frequently in many edge detection algorithms, has been shown to be feasibly implemented on digital hardware. However, the gradient magnitude of these implementations used the summation of the absolute values of the g_x and g_x gradients as its estimate $|\nabla f| \approx |g_x| + |g_y|$, whereas in our implementation, we used the actual square root operator to compute the gradient magnitude. Using the FRSR algorithm gives a more accurate estimate of the gradient magnitude as computed from the square root of the square of the gradients in both the horizontal and vertical directions $|\nabla f| = \sqrt{g_x{}^2 + g_y{}^2}$, compared with using the summation of the absolute values of the gradients.

5. ACKNOWLEDGEMENT

The authors are thankful to the Ministry of Higher Education of Malaysia for the award of the Fundamental Research Grant Scheme FRGS/1/2015/TK04/MMU/02/10 to support this project. We are also thankful to Jeannie Lau and Ang Boon Chong for the many technical discussions that helped us complete this project.

6. REFERENCES

Ananthalakshmi, A.V., Sudha, G.F., 2017. Design of a Reversible Floating-point Square Root using Modified Non-restoring Algorithm. *Microprocessors and Microsystems*, Volume 50, pp. 39–53

Arce, G.R., Bacca, J., Paredes, J.L., 2000. *Nonlinear Filtering for Image Analysis and Enhancement*. The Essential Guide to Image Processing. Massachusetts: Academic Press

Ashenden, P.J., 2008. *The Designer's Guide to VHDL*, Volume 3. Massachusetts: Morgan Kaufmann

Aubert, G., Kornprobst, P., 2006. *Mathematical Problems in Image Processing: Partial Differential Equations and the Calculus of Variations*. Volume 147. Berlin: Springer

Bertalmıo, M., Cheng, L.T., Osher, S., Sapiro, G., 2001. Variational Problems and Partial Differential Equations on Implicit Surfaces. *J. Computational Physics*, Volume 174(2), pp. 759–780

Bhagat, A.R., Dixit, S.R., Deshmukh, A.Y., 2015. VHDL Based Sobel Edge Detection. *International Journal Engineering Research and General Science*, Volume 3(1), pp. 1217–1223

Chambolle, A., 1994. Partial Differential Equations and Image Processing. *In:* Proc. IEEE Int. Conf. Image Processing, Volume 1, pp. 16–20

Ercegovac, M.D., Lang, T., Muller, J.M., Tisserand, A., 2000. Reciprocation, Square Root, Inverse Square Root, and Some Elementary Functions using Small Multipliers. *IEEE Transactions on Computers*, Volume 49(7), pp. 628–637

Ercegovac, M.D., Muller, J.M., Tisserand, A., 2005. Simple Seed Architectures for Reciprocal and Square Root Reciprocal. *In:* Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers, pp. 1167–1171

Istoan, M., Pasca, B., 2015. *Fixed-Point Implementations of the Reciprocal, Square Root and Reciprocal Square Root Functions*. Hal Archives-Ouvertes

Kanjar, D., Masilamani, V., 2013a. A New No-reference Image Quality Measure for Blurred Images in Spatial Domain. *International Journal of Image and Graphics*, Volume 1(1), pp. 39–42

Kanjar, D., Masilamani, V., 2013b. Image Sharpness Measure for Blurred Images in Frequency Domain. *Procedia Engineering*, Volume 64, pp. 149–158

Kanjar, D., Masilamani, V., 2017. Image Quality Assessment for Blurred Images using Nonsubsampled Contourlet Transform Features. *Journal of Computers*, Volume 12(2), pp. 156–164

Kho, D.C.K., Fauzi, M.F.A., Lim, S.L., 2018. Hardware Implementation of Low-latency 32-bit Floating-point Reciprocal Square Root. *Journal of Electrical & Electronic Systems*, Volueme 7(4), pp. 1–4

Kokkinos, I., 2013. *Introduction to Nonlinear Image Processing. Center for Visual Computing. Ecole Centrale Paris*. Available Online at http://vision.mas.ecp.fr/Personnel/iasonas/course/nonlinear.pdf

Kornprobst, P., Deriche, R., Aubert, G., 1999. Image Sequence Analysis via Partial Differential Equations. *Journal of Mathematical Imaging and Vision*, Volume 11(1), pp. 5–26

Kreyszig, E., 2011. *Advanced Engineering Mathematics*. 10th Edition. New Jersey: John Wiley & Sons

Lachowicz, S., 2008. Fast Evaluation of the Square Root and Other Nonlinear Functions in FPGA. *In:* Proceedings of the 4th IEEE International Symposium on Electronic Design, Test & Applications, pp. 474–477

Li, Y., Chu, W., 1997. Implementation of Single Precision Floating Point Square Root on FPGAs. *In:* Proceedings of the 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 226–232

Lomont, C., 2003. *Fast Inverse Square Root*. Technical Report 32. Department of Mathematics, Purdue University, West Lafayette, Indiana, USA

Mehra, R., Verma, R., 2012. Area Efficient FPGA Implementation of Sobel Edge Detector for Image Processing Applications. *International Journal of Computer Applications*, Volume 56(16), pp. 7–11

Mitra, S.K., Sicuranza, G. L., 2001. *Nonlinear Image Processing*. Massachusetts: Academic Press

Naik, S., Metkewar, P., 2015. Recognizing Offline Handwritten Mathematical Expressions (ME) based on a Predictive Approach of Segmentation using K-NN Classification. *International Journal of Technology*, Volume 6(3), pp. 345–354

Nanhe, A., Gawali, G., Ahire, S., Sivasankaran, K., 2013. Implementation of Fixed and Floating Point Square Root using Nonrestoring Algorithm on FPGA. *Int. J. Computer and Electrical Engineering*, Volume 5(5), pp. 533–537

Nelson, A.E., 2000. Implementation of Image Processing Algorithms on FPGA Hardware. *M.Sc. Dissertation*, Vanderbilt University, Nashville, TN, USA

Nosrat, A., Kavian, Y.S., 2012. Hardware Description of Multi-directional Fast Sobel Edge Detection Processor by VHDL for Implementing on FPGA. *International Journal of Computer Applications*, Volume 47(25), pp. 1–7

Pangestu, P., Gunawan, D., Hansun, S., 2017. Histogram Equalization Implementation in the Preprocessing Phase on Optical Character Recognition. *International Journal of Technology*, Volume 8(5), pp. 947–956

Robertson, M., 2012. A Brief History of InvSqrt. *B.Sc. Dissertation*, University of New Brunswick, New Brunswick, Canada

Sajid, I., Ahmed, M.M., Ziavras, S.G., 2010. Pipelined Implementation of Fixed Point Square Root in FPGA using Modified Non-restoring Algorithm. *In:* The 2nd International Conference Computer and Automation Engineering (ICCAE), Singapore, pp. 226–230

Sanduja, V., Patial, R., 2012. Sobel Edge Detection using Parallel Architecture based on FPGA. *International Journal Applied Information Systems (IJAIS)*, Volume 3(4), pp. 20–24

Singh, S., Saini, A.K., Saini, R., 2012. Real-time FPGA Based Implementation of Color Image Edge Detection. *International Journal of Image, Graphics, and Signal Processing*, Volume 4(12), pp. 19–25

Umar, A., Li, H., Aguirre, A., Zhu, Q., 2012. FPGA-based Reconfigurable Processor for Ultrafast Interlaced Ultrasound and Photoacoustic Imaging. *IEEE Transaction Ultrasonics, Ferroelectrics and Frequency Control*, Volume 59(7), pp. 1344–1353

Wang, X., 2007. Variable Precision Floating-Point Divide and Square Root for Efficient FPGA Implementation of Image and Signal Processing Algorithms. *Master's Thesis, Ph.D. Dissertation*, Northeastern Univ., Massachusetts, USA

Xu, W., Mueller, K., 2010. Evaluating Popular Non-linear Image Processing Filters for Their Use in Regularized Iterative CT. *In:* Proceedings of IEEE Nuclear Science Symposium (NSS/MIC), pp. 2864–2865

Yasri, I., Hamid, N.H., Yap, V.V., 2009. An FPGA Implementation of Gradient Based Edge Detection Algorithm Design. *In:* International Conference on Computer Technology and Development, Kota Kinabalu, Malaysia, Volume 2, pp. 165–169

Zafar, S., Adapa, R., 2014. Hardware Architecture Design and Mapping of Fast Inverse Square Root Algorithm. *In:* Proceedings 2014 International Conference on Advances in Electrical Engineering (ICAEE), pp. 1–4