Hindawi
International Journal of Reconfigurable Computing
Volume 2017, Article ID 1325493, 13 pages
https://doi.org/10.1155/2017/1325493

Research Article

Fuzzy Logic Based Hardware Accelerator with Partially Reconfigurable Defuzzification Stage for Image Edge Detection

Aous H. Kurdi, Janos L. Grantner, and Ikhlas M. Abdel-Qader

Electrical and Computer Engineering Department, Western Michigan University, Kalamazoo, MI 49009, USA

Correspondence should be addressed to Aous H. Kurdi; aoushammad.kurdi@wmich.edu

Received 5 December 2016; Accepted 8 February 2017; Published 1 March 2017

Academic Editor: Yuko Hara-Azumi

In this paper, the design and the implementation of a pipelined hardware accelerator based on a fuzzy logic approach for an edge detection system are presented. The fuzzy system comprises a preprocessing stage, a fuzzifier with four fuzzy inputs, an inference system with seven rules, and a defuzzification stage delivering a single crisp output, which represents the intensity value of a pixel in the output image. The hardware accelerator consists of seven stages with one clock cycle latency per stage. The defuzzification stage was implemented using three different defuzzification methods. These methods are the mean of maxima, the smallest of maxima, and the largest of maxima. The defuzzification modules are interchangeable while the system runs using partial reconfiguration design methodology. System development was carried out using Vivado High-Level Synthesis, Vivado Design Suite, Vivado Simulator, and a set of Xilinx 7000 FPGA devices. Depending upon the speed grade of the device that is employed, the system can operate at a frequency range from 83 MHz to 125 MHz. Its peak performance is up to 58 high definition frames per second. A comparison of this system's performance and its software counterpart shows a significant speedup in the magnitude of hundred thousand times.

1. Introduction

Digital system design using Field Programmable Gate Arrays (FPGAs) focuses on performance, device utilization, and rapid development. Xilinx Vivado HLS offers a great development environment that enables the analysis of the system's performance and design optimization. It also facilitates modularized system design. The Vivado Design Suite provides the means for developing dynamic, partially reconfigurable designs in which different hardware modules can be swapped in and out to utilize available hardware resources on the fly. FPGAs, as a platform, represent one of the most qualified contenders for hardware implementation of digital signal processing systems [1]. In the Xilinx 7 Series devices, the programmable elements organized in blocks called Configurable Logic Block (CLB). Each CLB is comprised of two slices, and each slice is provided with a 6-input 1-output look-up table (LUT), distributed memory, shift register, high-speed logic for arithmetic functionality, a wide multiplexer, and a switching matrix to facilitate the access to routing elements on the chip [2]. The synthesizer tool assigns the chip's resources, mainly the CLBs, in accordance with the designer choice to implement sequential or combinational logic circuits.

Digital systems by nature rely on Boolean logic. Boolean logic has been, conventionally, the staple of knowledge representation for quite long. The main shortcoming in this regard is the incomplete applicability to situations of uncertainty and inaccuracy [3]. Thus, conventional approaches founded on Boolean logic do not provide suitable frameworks to represent human knowledge that is characterized by the uncertainty and fuzziness associated with the human cognitive function. Fuzzy logic, however, provides a mathematically feasible framework to represent degrees of truth and falsehood in contrast to the classic true or false values of Boolean logic. Fuzzy logic is referred to here in the wide-sense [4] that includes the concept of fuzzy sets [5] and approximate reasoning. For many applications, fuzzy logic has become an indispensable tool. Those applications include system control, intelligent systems, and image processing [6].

Most image processing algorithms contain edge detection as a vital part. Edge detection is, essentially, any method or

algorithm that determines the set or sets of points within a digital image at which the gradient of intensity becomes a rapidly increasing or decreasing function of spatial coordinates [7, 8]. These points are grouped into curved line shapes called edges. Different methods have been developed to extract the edges in an image such as the Sobel operator, Laplacian, and Prewitt. These methods use specific parameters, such as a threshold, to complete the edge detection process [9].

Fuzzy logic based edge detection makes use of human knowledge to identify edges. A proposal for an improved edge detection algorithm using fuzzy logic was presented in [10]. The authors therein applied fuzzy logic techniques on a 3 × 3 pixels' mask. That mask is exploited in the process of examining each pixel's relation to its neighbor pixels. Each pixel is considered as a fuzzy input resulting in a multi-input-single-output (MISO) fuzzy system. Another approach used the pixels' gradient and standard deviation values as inputs to the fuzzy system [11]. In this paper, the suggested system works in such a way that it decides on which pixel is considered an edge, or not, by carrying out inference calculations utilizing a set of fuzzy IF-THEN rules. Using fuzzy logic to construct an edge detection algorithm would have the obvious potential. It could incorporate human knowledge and intuition into a model that can adapt to a substantial departure from the expected input images, such as the presence of noise in the input image, rather than using only a single real number to substantiate detection, that is, using a static threshold value. Fuzzy logic based systems work with a linguistic representation of knowledge in a way that describes uncertainty in the form of IF-THEN rules. Complex systems can be modeled using those rules that are intuitively recognizable to human beings [12]. To unlock the potential of fuzzy logic based systems in real life applications, practical platforms with low energy consumption and high computing power are crucially needed to implement them. Authors in [13], worked on utilizing FPGAs in implementing a fuzzy logic controller to track the maximum power point of the photovoltaic system. The authors used the Very High-Speed Hardware Description Language (VHDL) to design and implement a Mamdani-type [13] fuzzy controller.

The partial reconfiguration design approach aims to maximize device utilization efficiency by allowing different functional modules to use a specified set or sets of device resources called reconfigurable partition (PR) in interchangeable fashion while preserving the functionality of the rest of the system. In [14], a proposal for real-time tasks scheduling using partially reconfigurable FPGA design was discussed. The authors suggested dividing the FPGA fabric floor into homogenous blocks. Their experimental outcomes showed high resource utilization efficiency.

The rest of the paper is structured as follows: Section 2 introduces the proposed fuzzy edge detector system. The architecture of the hardware accelerator is described in Section 3. In Section 4, the dynamic partial reconfiguration design component of the edge detector system is discussed. The evaluation of the results is presented in Section 5. Conclusions are given Section 6.

2. The Proposed Fuzzy Edge Detector System

The proposed edge detector is a Mamdani-type [15] fuzzy system with four fuzzy inputs, one crisp output, and a knowledge base made of seven IF-THEN rules. It is important to note that in a Mamdani-type fuzzy system no implication functions [16] are used to create the fuzzy knowledge base representation. The first two inputs are the gradients on the x-axis and the y-axis out of a kernel of 3 × 3 pixels. GX and GY are the associated fuzzy sets, respectively. The third input is the output of a low-pass filter, and the corresponding fuzzy set is named LF, while the final input is the output of a high-pass filter, and the corresponding fuzzy input is referred to by HF.

2.1. Preprocessing. In preprocessing, a kernel of 3 × 3 pixels formed from the input image is used to compute the gradients in x-direction and y-directions, low-pass filter, and high-pass filter using (1), (2), (3), and (4), respectively. $I(x, y)$ stands for the kernel with the targeted pixel at the center.

$$GX = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \cdot I(x, y), \tag{1}$$

$$GY = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \cdot I(x, y), \tag{2}$$

$$\mathrm{LP} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \cdot \frac{1}{9} \cdot I(x, y), \tag{3}$$

$$\mathrm{HP} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \cdot \frac{1}{9} \cdot I(x, y). \tag{4}$$

2.2. Fuzzification. For fuzzification, crisp inputs are transformed into fuzzy inputs. Linguistic variables were created to represent the fuzzy qualities over a practical range of crisp values. For each linguistic variable three fuzzy sets were defined over the universal sets of discourse, namely, LOW, MED, and HIGH.

2.3. The Inference System. For each input variable, three membership functions (MF) were defined, LOW, MED, and HIGH. LOW and HIGH are trapezoid-shaped MFs while MED is a triangle-shaped one. The membership functions are distributed over the universal set of discourse ranging from 0 to 255 (the intensity range in a grayscale image) as illustrated in Figure 1.

The system features a Mamdani inference system [17]. The knowledge base is made up of seven fuzzy IF-THEN rules as shown in Table 1. All the rules are assigned the weight value of 1. Fuzzy intersection and union, in the implication process, are represented by MIN and MAX operators, respectively.

Table 1: The fuzzy rules.

Fuzzy inputs				Fuzzy output
GX	*GY*	HP	LP	*E*
LOW	LOW	x	x	LOW
MED	MED	x	x	MED
HIGH	HIGH	x	x	HIGH
LOW	x	LOW	x	MED
x	LOW	LOW	x	MED
LOW	x	x	LOW	LOW
x	LOW	x	LOW	LOW

2.4. Defuzzification. For the single output variable *E* (Edge), three membership functions were defined: LOW, MED, and HIGH. All those membership functions were chosen to be triangular as shown in Figure 2. They are distributed over the universal set of discourse ranging over values from 0 to 255 (the intensity range of a pixel in the edges output image). Three defuzzification methods were implemented to map the fuzzy output into a crisp value. These methods are the smallest of maxima (SOM), the mean of maxima (MOM), and the largest of maxima (LOM).

3. The Architecture of the Pipelined Hardware Accelerator

In this research, a pipelined hardware accelerator was designed and implemented on Xilinx 7000-series devices using Vivado HLS and Vivado Design Suite. The system consists of four main units: preprocessor, fuzzifier, inference system, and defuzzifier. They have implemented over seven pipeline stages. The latency of each stage is a single clock cycle. The preprocessor uses three stages; the inference system employs two stages while the other two blocks use one stage each. The defuzzifier unit was implemented using Vivado HLS while the rest were implemented as Register Transfer Level (RTL) design using VHDL in the Vivado Design Suite.

3.1. Preprocessing Unit. Figure 3 illustrates the preprocessor unit's functional block diagram. The hardware architecture of the preprocessor consists of a Block RAM (BRAM) Module that is configured as a Dual-Port RAM with asynchronous Read and Writes cycles. The memory organization is set up with a parallel data width of 72 bits by 512 locations. Each location is assigned the representation of the intensity of 9 pixels forming a 3×3 kernel window. The first stage of the pipeline reads one location from memory. Dedicated blocks of hardware were implemented for computing the gradients in the x- and the y-directions, the low-pass filter, and the high-pass filter, respectively. Each block employs two stages of the system's seven pipeline stages, working in parallel.

For *GX* and *GY*, two specialized subblocks were designed to execute the process of computing the positive and negative parts of the gradient masks as in (1) and (2). These subblocks work in parallel forming the second stage of the pipeline. The third stage performs the tasks as follows: addition of the outputs of the previous subblocks, followed by finding the magnitude of the sum and then scaling that down to the established maximum value (255 in this case) if the results surpassed that maximum.

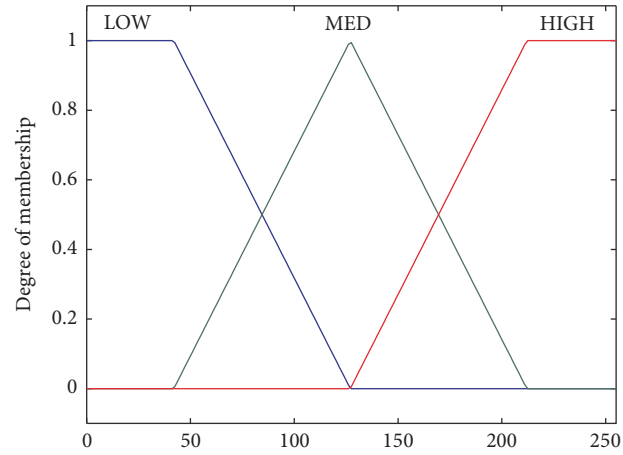

Figure 1: The membership functions of input *GX*, *GY*, HP, and LP.

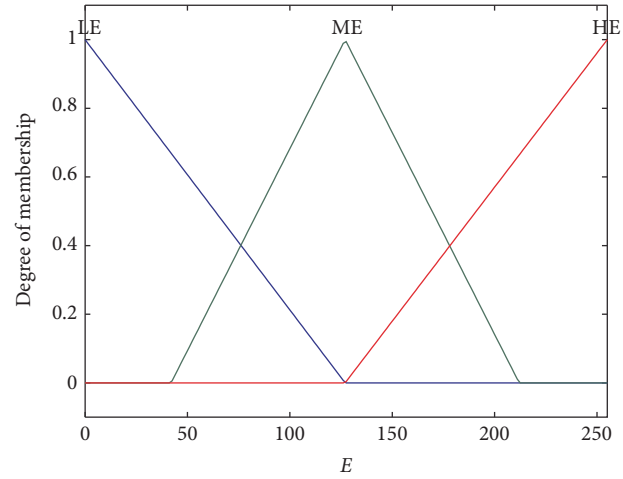

Figure 2: Membership functions for output *E*.

For LF, three subblocks were designed to find the sum of each row as in (3). These subblocks work in parallel and occupy the second stage of the pipeline in the LP block. The third stage computes the sum of the results of the previous stage and divides it by 9 to find the average.

The last block is the HF that is also divided into two stages. The first one is composed of three subblocks. The first subblock performs multiplication of the first four elements of the HP mask as in (4) with their corresponding input pixels and calculates the sum. The second subblock does the same as the first one, but for the last four elements of the HF mask. The third subblock carries out the multiplication of the center element by 8. The second stage computes the sum of the previous stage and then divides it by 9.

3.2. Fuzzification Unit. The fourth stage of the hardware accelerator pipeline is the fuzzification unit. This unit performs the transformation of the crisp inputs into fuzzy

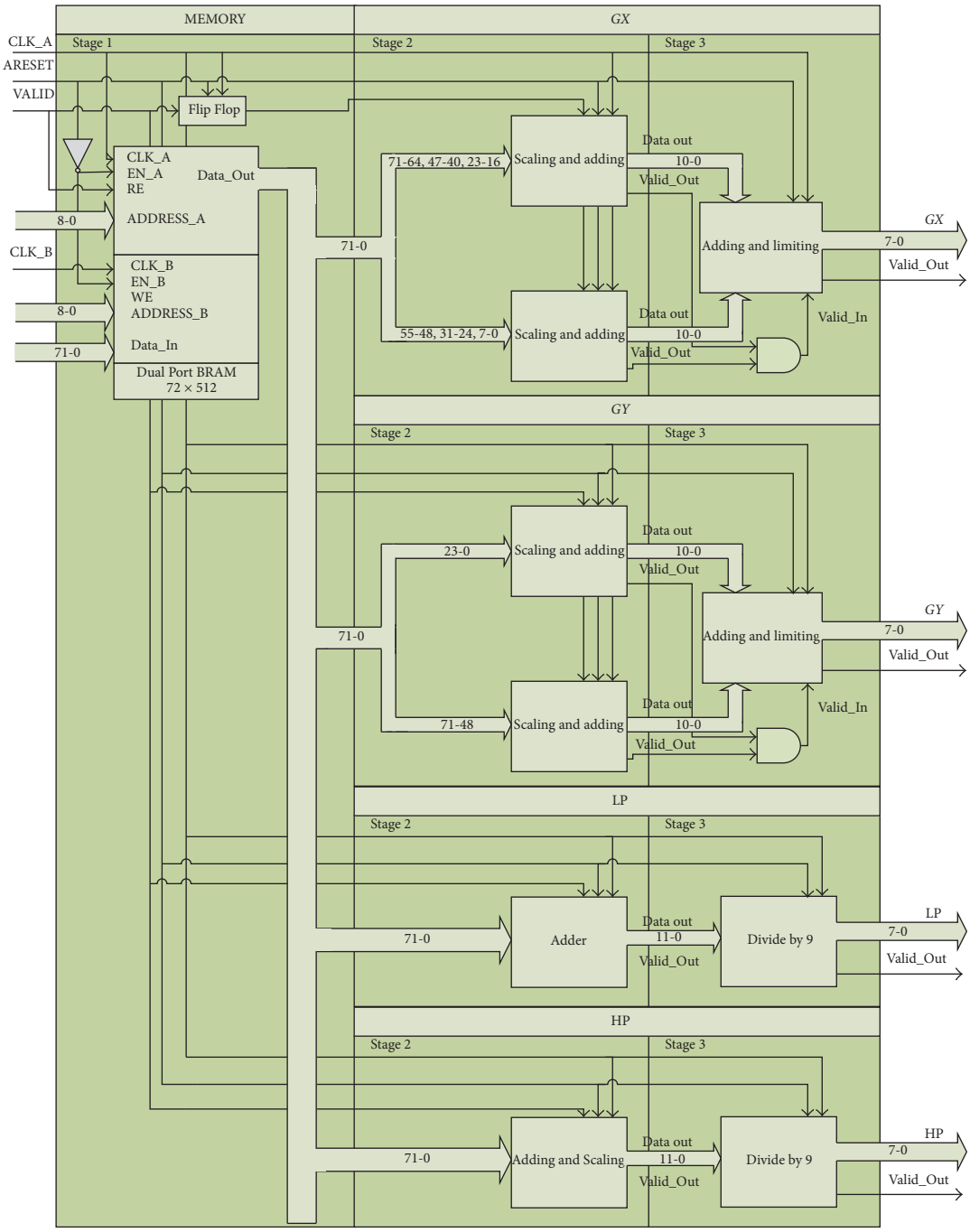

Figure 3: Preprocessing unit functional block diagram.

variables. The fuzzifier unit's functional block diagram is illustrated in Figure 4. The fuzzifier block consists of four identical subblocks working in parallel. These subblocks map the crisp inputs to linguistic labels in the corresponding fuzzy universal sets of discourse along with the degrees of consistency.

The inputs of the fuzzifier block are *GX*, *GY*, LP, and HP, which are the outputs of the preprocessor block, and control signals such as CLK, ARESET, and Valid_In. The outputs of the fuzzifier block are as follows: fuzzy variable *GX* (*GX*_VF), degree of consistency of variable *GX* (*GX*_DoM), fuzzy variable *GY* (*GY*_FV), degree of consistency of variable *GY* (*GY*-DoM), fuzzy variable LP (LP_FV), degree of consistency of variable LP (LP_DoM), fuzzy variable HP (HP_VF), and degree of consistency of variable HP (HP_DoM), and Valid_Out.

3.3. Inference System Unit. The hardware design of the inference system utilizes two stages of the system's seven pipeline stages. The first stage implements the knowledge base, which

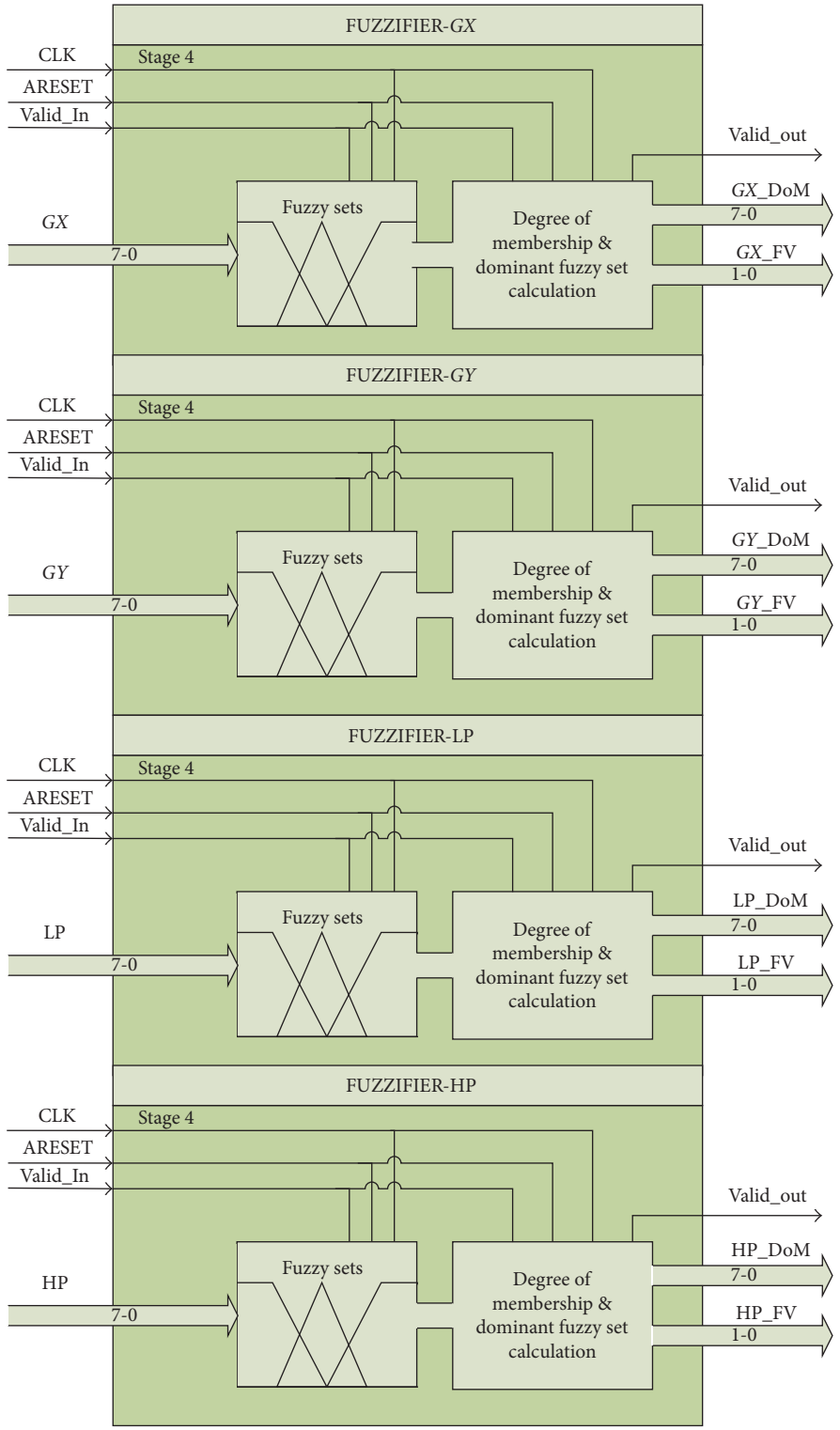

Figure 4: Fuzzification unit functional block diagram.

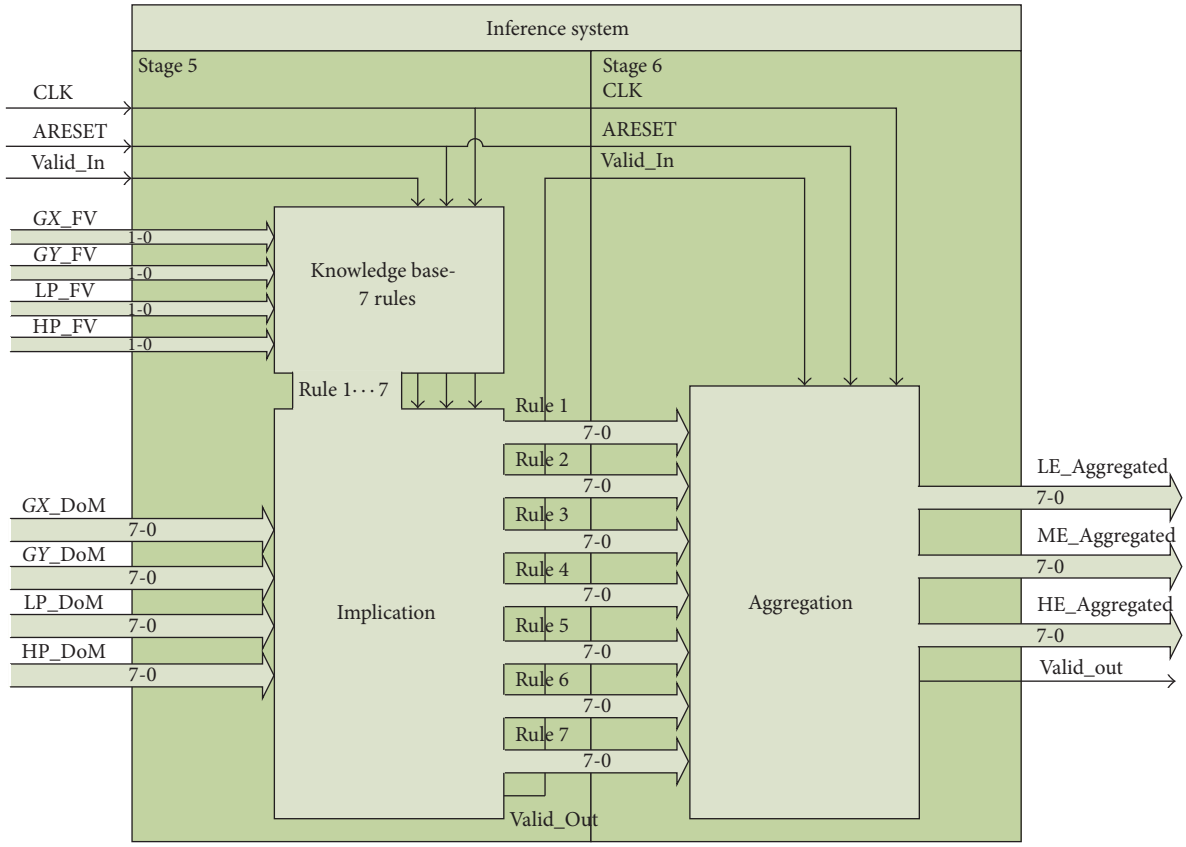

Figure 5: Inference system unit functional block diagram.

is composed of seven IF-THEN rules and computes the implication for each rule. The second stage completes aggregating the outcomes of the rules into three fuzzy variables LE, ME, and HE, respectively. The schematic Figure 5 represents the inference system unit functional block diagram.

3.4. Defuzzification Unit. In this unit, the aggregated fuzzy variables LE, ME, and HE are defuzzified using SOM, or MOM, or LOM defuzzification methods. The output of the defuzzification unit is an 8-bit representation of pixel intensity values in the output image. The defuzzification unit's block diagram is presented in Figure 6.

The defuzzification algorithms were implemented using Vivado HLS. The implementation of the defuzzification modules was written using C code along with a test bench. Vivado HLS synthesized the top-level function in the C code, named rModule_Defuzzification, into RTL design. Arbitrary precision integer of the length of 8 was used to implement the function interface. Pseudocode 1 shows the pseudocode for the implemented design.

4. Dynamic Partially Reconfigurable Design

At the time of writing this paper, the Vivado Design Suite supports only nonproject Tcl-based design flow for partial reconfiguration designs on FPGAs using bottom-up synthesis. The bottom-up approach uses multiple netlists from different projects or design check points (DCP) to create the static design along with the reconfigurable partitions (RP). The static design includes all the logic that is not subject of reconfiguration. RP is a design element that is marked for reconfiguration. The portion of the design that will occupy the RP is known as reconfigurable module (RM) [18]. The process of generating a partially reconfigurable design from RTL to partial Bitstream generation using the Vivado Design Suite can be summarized as follows:

(1) Synthesize the static design and generate DCP.

(2) Synthesize the RM and generate DCP using bottom-up methodology.

(3) The DCP created before will be used to load the static design along with RM for each RP.

(4) Use the floor-planning tool to define each RP.

(5) Design rule checker (DRC) will be executed to verify the correctness of the floor-planning process.

(6) Optimize, place, and route the design and save DCP.

(7) Reiterate steps (3) to (6) for each single RM.

(8) Execute the PR_Verify command to validate the reconfigurable design.

(9) Generate partial Bitstream files for each single RM.

The partially reconfigurable design of the hardware accelerator implements the first six stages as a static portion while

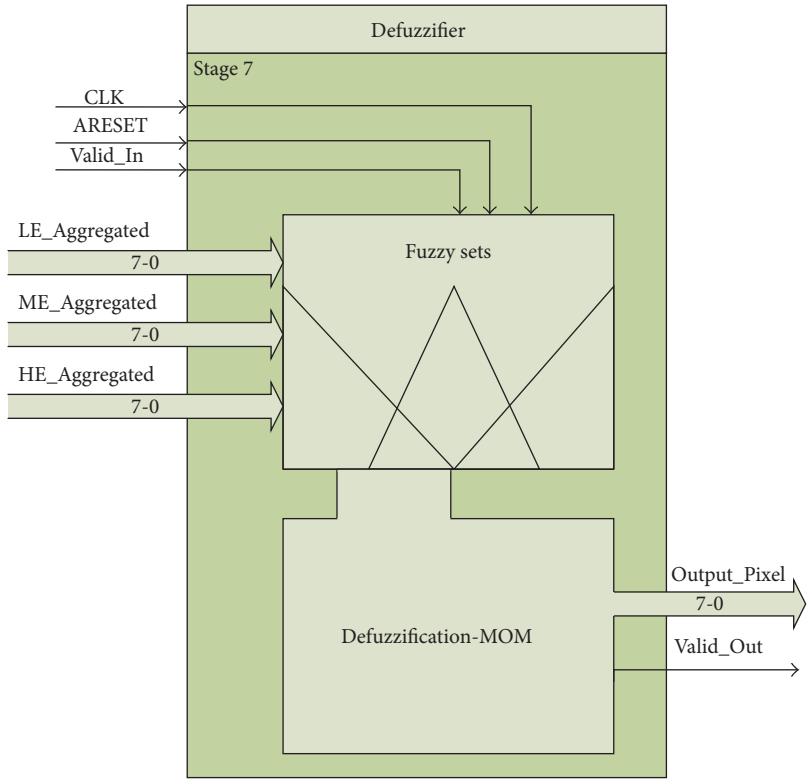

Figure 6: Defuzzification unit functional block diagram.

```
rModule_Defuzzification (uint8 LE, uint8 ME, unit8 HE, uint8 Output)
{
    If (LE > ME and LE> HE)
        Output= Calculate_Crisp (MF, LE)
    Else If (ME > LE and ME> HE)
        Output= Calculate_Crisp (MF, ME)
    Else If (HE > LE and HE> ME)
        Output= Calculate_Crisp (MF, HE)
}
Uint8 Calculate_Crisp (MembershipFunction MF, uint8 Value)
{
    If (MF is Triangle)
        Return Triangle_Crisp(Value)
    else
        Return Trapizoid_Crisp(Value)
}
```

Pseudocode 1: Pseudocode for SOM defuzzification.

the defuzzification unit uses one RP and three RM modules as illustrated in Figure 7.

5. Evaluation of the Results

In this research, a hardware accelerator for fuzzy logic based edge detector with partially a reconfigurable defuzzification unit was designed, implemented, and tested. The hardware architecture was developed using Xilinx Vivado HLS and Vivado Design Suite. The performance of the hardware accelerator was investigated using a set of Xilinx Artix7 and Kintex7 devices. The fuzzy system's performance has been compared, in the presence of noise, to three traditional edge detection techniques.

5.1. The Performance of the Hardware Accelerator. The hardware accelerator design is based upon a pipeline architecture of seven stages. Each stage requires one clock cycle of execution time. The system was tested using three Xilinx Artix7 and three Xilinx Kintex7 devices with different speed

Table 2: Device utilization and maximum operating speed.

Device name	Maximum speed (MHz)	BRAM	DSP48E	FF	LUT
Artix7-Xc7a100t csg324-1	88.333	1	2	269	843
Artix7 Xc7a100t csg324-2	90.909	1	2	269	843
Artix7 Xc7a100t csg324-3	100.00	1	2	269	843
Kintex7 Xc7k160t fbg484-1	100.00	1	2	269	843
Kintex7 Xc7k160 tfbg484-2	111.111	1	2	269	843
Kintex7 Xc7k160 tfbg484-3	125.00	1	2	269	843

Table 3: Device utilization and maximum operating speed of (SOM) defuzzification module.

Design	Hardware		Timing				Utilization			
Design method	Device family	Device name	Targeted frequency (MHz)	Targeted time (ns)	Estimated time (ns)	Number of cycles required	BRAM	DSP48E	FF	LUT
Nonoptimized HLS	Artix-7	Xc7a100tcsg324-1	88.333	12	9.79	1–12	0	0	108	178
Optimized HLS	Artix-7	Xc7a100tcsg324-1	88.333	12	10.42	1	0	2	0	55
Optimized HLS	Artix-7	Xc7a100tcsg324-2	90.909	11	9.16	1	0	2	0	55
Optimized HLS	Artix-7	Xc7a100tcsg324-3	100.00	10	8.28	1	0	2	0	55
Optimized HLS	Kintex-7	Xc7k160tfbg484-1	100.00	10	8.21	1	0	2	0	55
Optimized HLS	Kintex-7	Xc7k160tfbg484-2	111.111	9	7.22	1	0	2	0	55
Optimized HLS	Kintex-7	Xc7k160tfbg484-3	125.00	8	6.56	1	0	2	0	55
RTL	Artix-7	Xc7a100tcsg324-1	88.333	12	11.07	1	0	0	9	65
RTL	Artix-7	Xc7a100tcsg324-2	90.909	11	9.87	1	0	0	9	65
RTL	Artix-7	Xc7a100tcsg324-3	100.00	10	8.91	1	0	0	9	65
RTL	Kintex-7	Xc7k160tfbg484-1	100.00	10	8.73	1	0	0	9	65
RTL	Kintex-7	Xc7k160tfbg484-2	111.111	9	7.51	1	0	0	9	65
RTL	Kintex-7	Xc7k160tfbg484-3	125.00	8	6.76	1	0	0	9	65

grades as indicated in Table 2. The tests showed that the system could work with a clock frequency rate of up to 125 MHz, hence, producing an output pixel in every eight ns.

Vivado HLS was used to implement the defuzzification unit. Two design approaches were investigated, the none-optimized design and the inline function optimization. The none-optimized design approach produces an RTL design that requires a latency of 1~12 cycles. The inline function optimization design utilizes DSP48E slices to perform the mathematical operation required to calculate crisp values. The execution time was optimized to just one clock cycle. The optimized approached showed better utilization and execution times compared to the hard-coded RTL design. The hardware utilization, timing, and design approach for the defuzzification unit using HLS and RTL on different devices are illustrated in Table 3. The timing simulations for the nonoptimized design and the inline function optimization for the SOM defuzzification module are shown in Figures 8 and 9, respectively.

Simulation results in Figure 10 show the system operating with an 11 ns clock cycle. The system needs 77 ns (7 cycles) to fill up the pipeline and after those another 176 ns (16 cycles) to produce 16 outputs as illustrated. To compare the performance of the hardware accelerator to its software

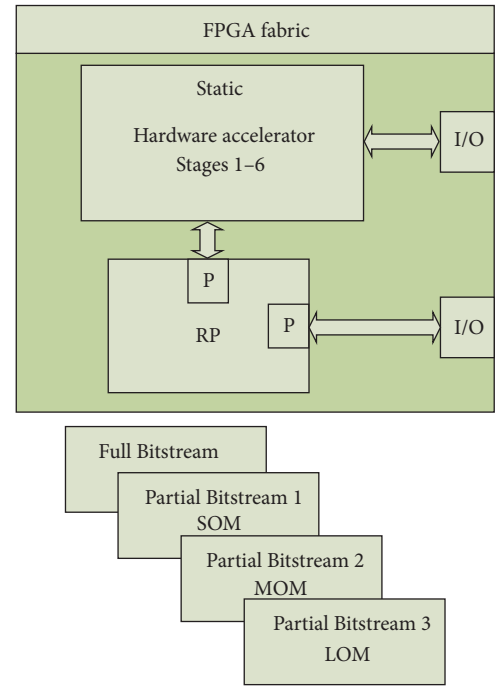

Figure 7: Partially reconfigurable system diagram.

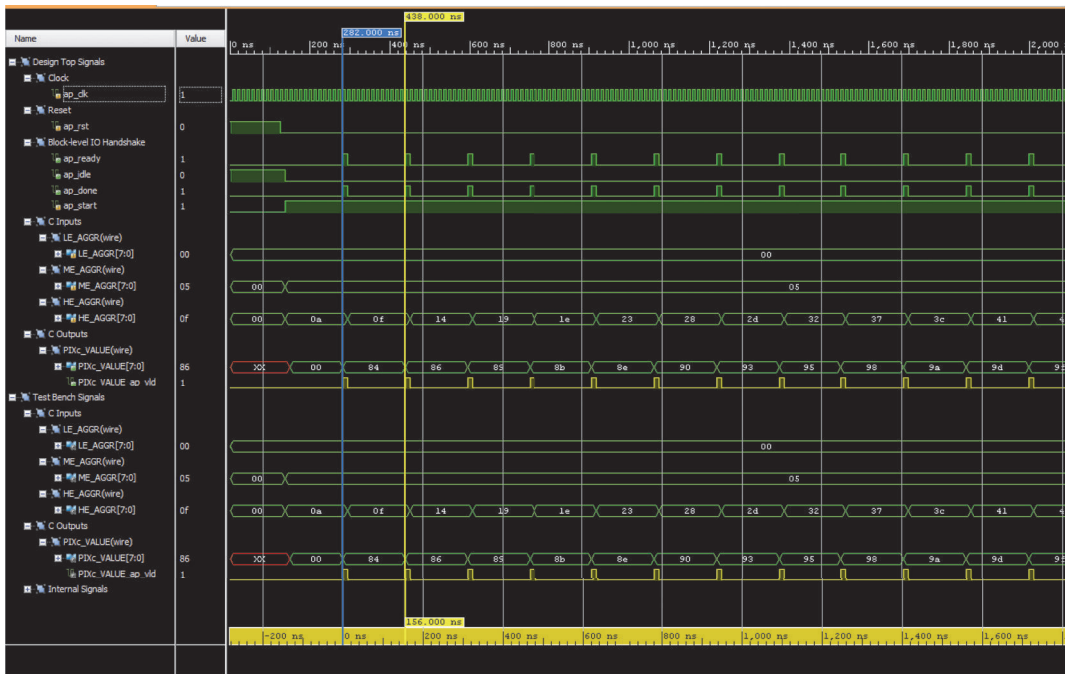

Figure 8: Simulation of the defuzzification unit using nonoptimized approach.

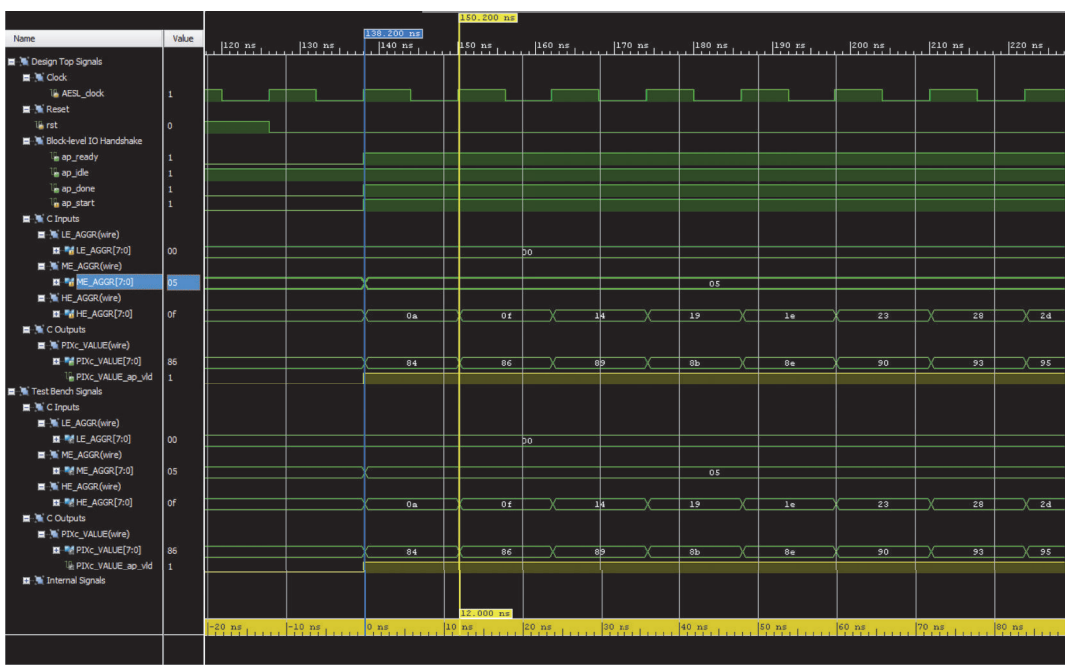

Figure 9: Simulation of the defuzzification unit using the inline function optimization.

counterpart, MATLAB was used to implement the system on a PC with Intel Core i7 processor and 8 GB of memory. The tests yield an execution time of 1.3178 milliseconds per pixel.

5.2. The Performance of the Proposed Fuzzy System. The proposed system was tested using two benchmark grayscale images, the cameraman and Lena. The performance results of the systems were also compared to other edge detection techniques such as Sobel's, Roberts's, and the Marr-Hildreth edge detection methods choosing Signal to Noise Ratio (SNR) as a quantitative measure. SNR, which is computed using (5), is the most widely used nondimensional parameter

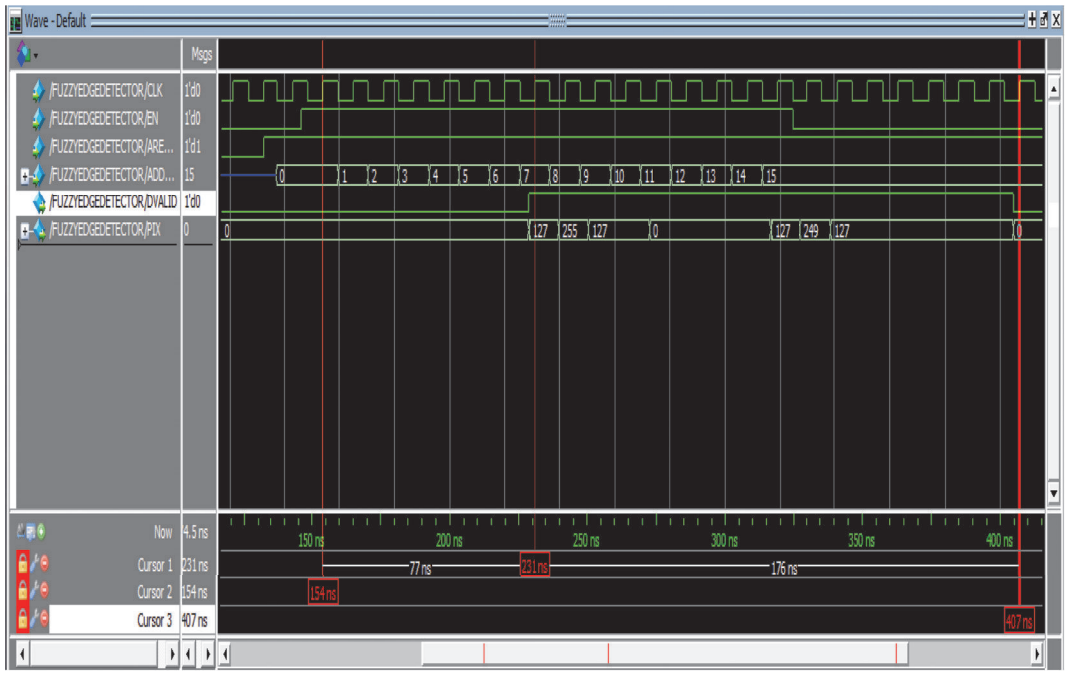

Figure 10: System simulation on Artix-7 Xc7a100tcsg324-2.

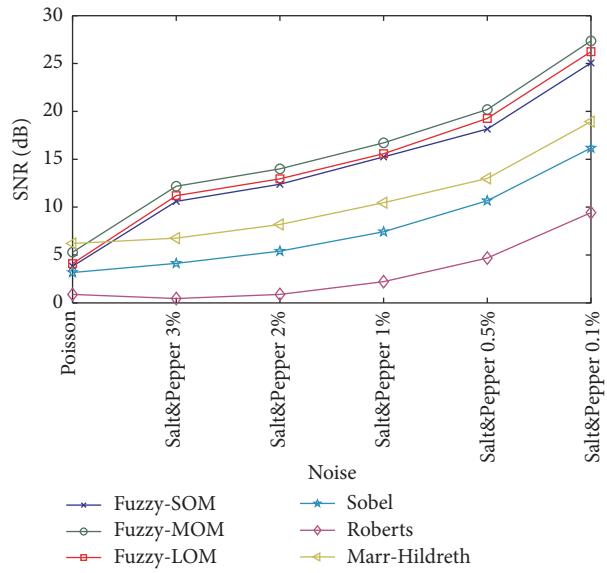

Figure 11: SNR in dB using different artificial noise (cameraman's image as source).

of the imaging system for measuring its sensitivity to noise [19].

$$\mathrm{SNR} = 10\log_{10}\frac{\sum_{0}^{n_x-1}\sum_{0}^{n_y-1}\left[r(x,y)\right]^2}{\sum_{0}^{n_x-1}\sum_{0}^{n_y-1}\left[r(x,y)-t(x,y)\right]^2}. \quad (5)$$

In (5), r and t stand for the respective grayscale intensities in the reference image and the image to be tested at pixel (x, y) in images of n_x by n_y size. The proposed system shows comparable or better immunity to noise compared to the other methods except the case with Poisson noise where Marr-Hildreth algorithm showed a better performance. The noise test was performed using artificial noise added to the original image as shown Figures 11–14.

6. Conclusions

A seven-stage pipeline architecture for fuzzy logic edge detector with partially reconfigurable defuzzifier was proposed and implemented using Xilinx Vivado HLS and Xilinx Vivado Design Suite. Three Xilinx Artix7 and three Kintex7 devices were used to evaluate the system performance. The

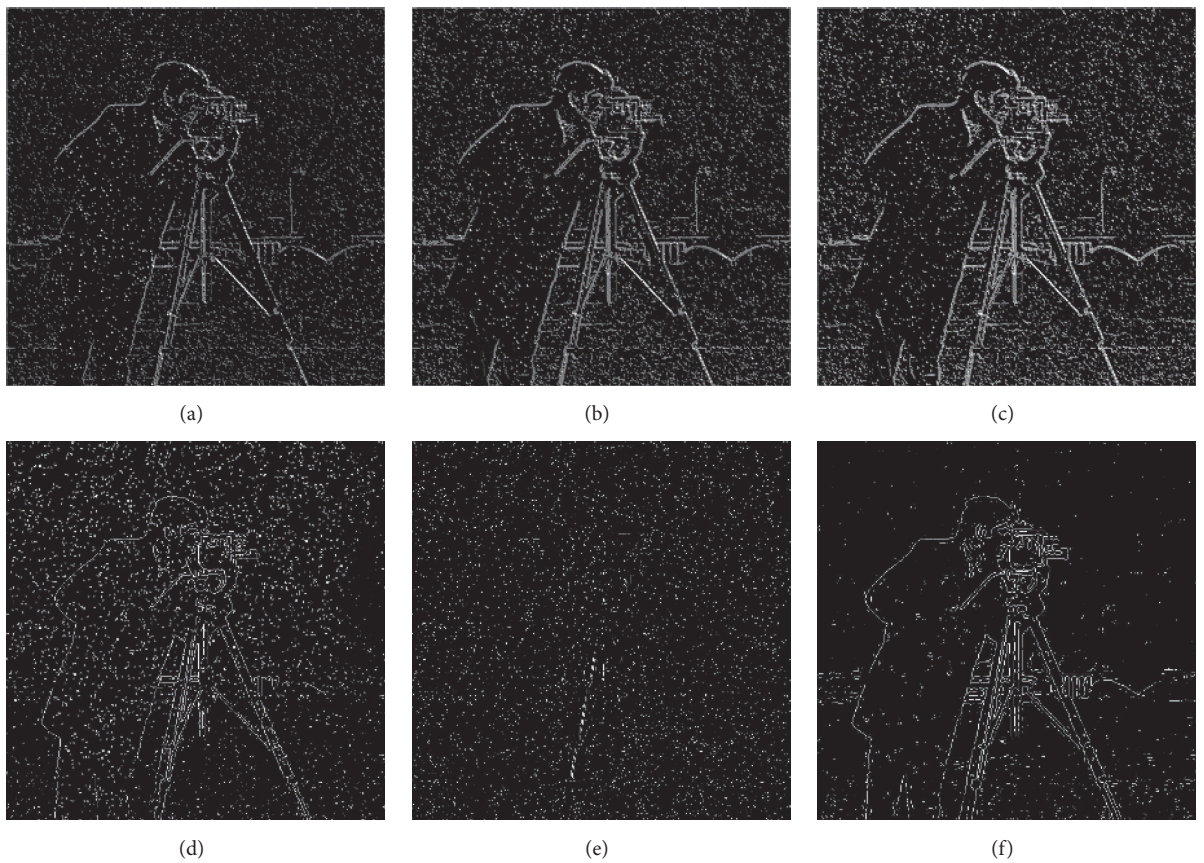

Figure 12: Cameraman with 3% salt and pepper: (a) SOM, (b) MOM, (c) LOM, (d) Sobel, (e) Roberts, and (f) Marr-Hildreth.

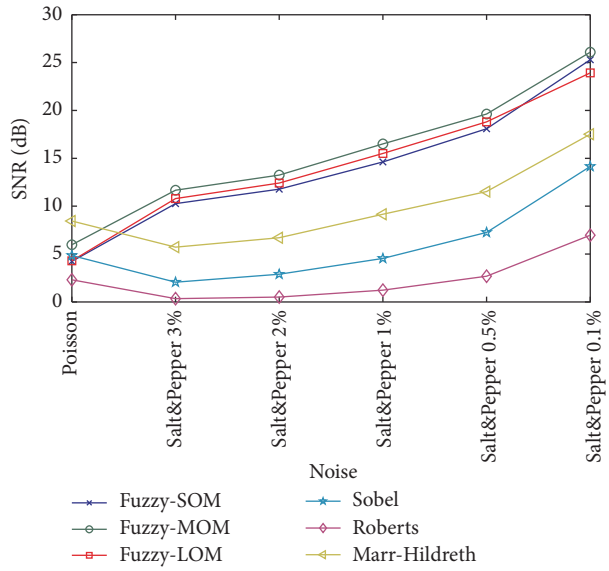

Figure 13: SNR in dB using different artificial noise (Lena's image as source).

performance of the hardware accelerator was compared to a functionally equivalent software counterpart and the results have shown a speedup of 109,618 times. The system's noise immunity was compared with other traditional methods using SNR as a quantitative performance measure. The system delivers better results in most of the comparisons. The defuzzification unit was implemented and optimized using Vivado HLS. The RTL code generated by HLS showed better performance and device utilization as compared to a hard-coded RTL that was generated by Vivado Design Suite using

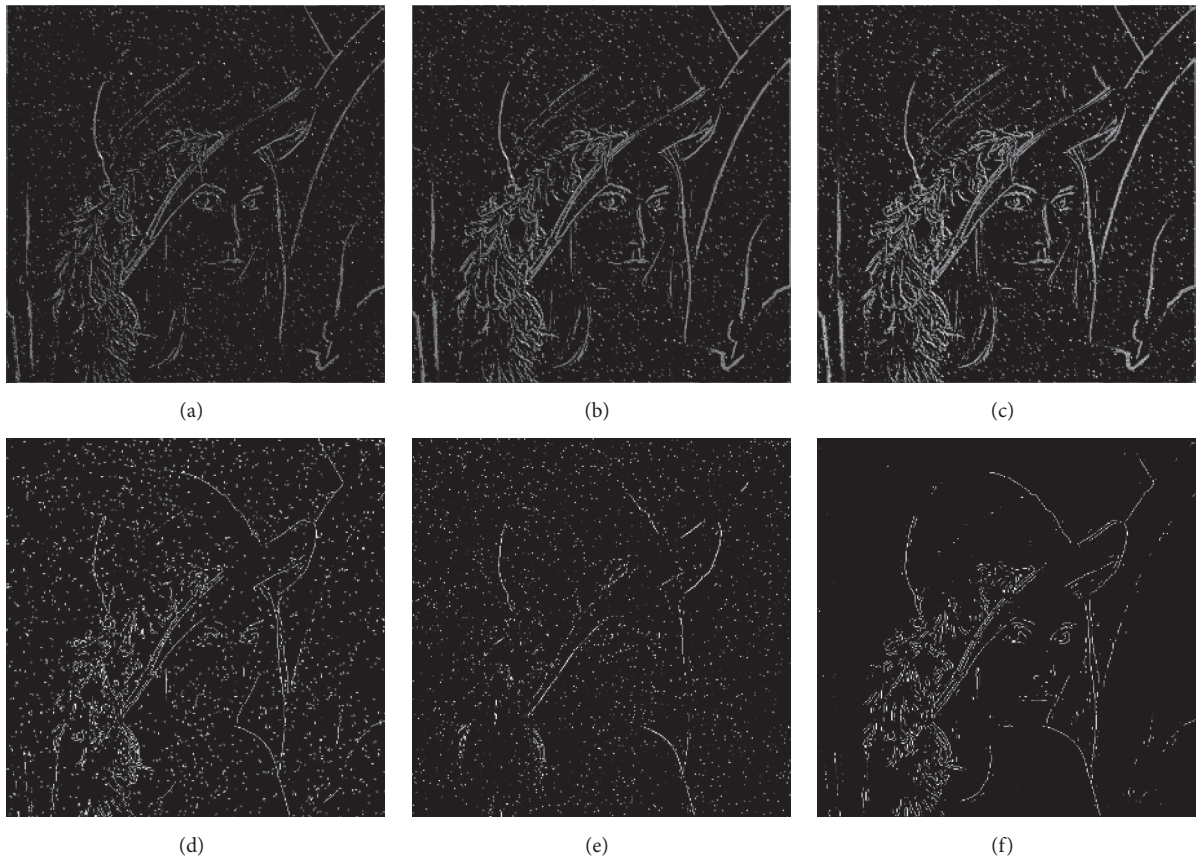

Figure 14: Lena with 1% salt and pepper: (a) SOM, (b) MOM, (c) LOM, (d) Sobel, (e) Roberts, and (f) Marr-Hildreth.

VHDL. Working with a system clock of 125 MHz, the system processed up to 58 HD frames per second. By attaching suitable input and output peripheral devices to the proposed hardware accelerator, it will make a powerful device for real-time applications.

Disclosure

Aous H. Kurdi was with Computer and Software Engineering Department, University of Technology, Baghdad, Iraq.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] C. Dick and F. Harris, "FPGA signal processing using Sigma-Delta modulation," *IEEE Signal Processing Magazine*, vol. 17, no. 1, pp. 20–35, 2000.

[2] Xilinx, "7 Series FPGAs Configurable Logic Block: User Guide. UG474 (v1.8)," September 2016.

[3] L. A. Zadeh, "Knowledge representation in fuzzy logic," *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, no. 1, pp. 89–100, 1989.

[4] L. A. Zadeh, "What is Fuzzy Logic?" January 2013.

[5] L. A. Zadeh, "Fuzzy sets," *Information and Computation*, vol. 8, no. 3, pp. 338–353, 1965.

[6] K. Tanaka, *An Introduction to Fuzzy Logic for Practical Applications*, Springer, 1997.

[7] S. E. Umbaugh, *Digital Image Processing and Analysis: Human and Computer Vision Applications with CVIPtools*, CRC Press, 2016.

[8] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.

[9] Y. Becerikli and T. M. Karan, "A new fuzzy approach for edge detection," in *Proceedings of the 8th International Workshop on Artificial Neural Networks (IWANN '05): Computational Intelligence and Bioinspired Systems*, pp. 943–951, Springer, Vilanova i la Geltrú, Spain, June 2005.

[10] D. N. Oliveira, A. P. De Souza Braga, and O. Da Mota Almeida, "Fuzzy logic controller implementation on a FPGA using VHDL," in *Proceedings of the Annual North American Fuzzy Information Processing Society Conference (NAFIPS '10)*, 6, 1 pages, July 2010.

[11] A. D. Borkar and M. Atulkar, "Detection of edges using fuzzy inference system," *The International Journal of Innovative Research in Computer and Communication Engineering*, vol. 1, no. 1, 2013.

[12] F. A. Tab and O.-K. Shahryari, "Fuzzy edge detection based on pixel's gradient and standard deviation values," in *Proceedings of the International Multiconference on Computer Science and*

Information Technology (IMCSIT '09), pp. 7–10, IEEE, October 2009.

[13] L. Bouselham, M. Hajji, B. Hajji, A. E. Mehdi, and H. Hajji, "Hardware implementation of fuzzy logic MPPT controller on a FPGA platform," in *Proceedings of the 3rd International Renewable and Sustainable Energy Conference (IRSEC '15)*, pp. 1–6, IEEE, Marrakech, Morocco, December 2015.

[14] S. Saha, A. Sarkar, and A. Chakrabarti, "Scheduling dynamic hard real-time task sets on fully and partially reconfigurable platforms," *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 23–26, 2015.

[15] E. H. Mamdani and S. Assilian, "Experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, vol. 7, no. 1, pp. 1–13, 1975.

[16] W. Bandler and L. J. Kohout, "Semantics of implication operators and fuzzy relational products," *International Journal of Man-Machine Studies*, vol. 12, no. 1, pp. 89–116, 1980.

[17] E. H. Mamdani, "Application of fuzzy logic to approximate reasoning using linguistic synthesis," *IEEE Transactions on Computers*, vol. C-26, no. 12, pp. 1182–1191, 1977.

[18] Xilinx, Vivado Design Suite User Guide Partial Reconfiguration UG909 (v2016.1), April 2016.

[19] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, Pearson, New York, NY, USA, 2008.