

Fast Double-Parallel image processing based on FPGA

Ye Li*, Qingming Yao[†], Bin Tian[‡], and Wencong Xu[§]

*State Key Laboratory of Intelligent Control and Management of Complex Systems
Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
Email: ye.li@ia.ac.cn

[†]State Key Laboratory of Intelligent Control and Management of Complex Systems
Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
Email: qingming.yao@ia.ac.cn

[‡]State Key Laboratory of Intelligent Control and Management of Complex Systems
Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
Email: bin.tian@ia.ac.cn

[§]School of Mechanical, Electrical and Information Engineering
Shandong University, Weihai 264209, China
Email: wencong.xu@gmail.com

Abstract—Due to FPGA's flexibility and parallelism, it is popular for accelerating image processing. In this paper, a double-parallel architecture based on FPGA has been exploited to speed up median filter and edge detection tasks, which are essential steps during image processing. The double-parallel scheme includes an image-level parallel and an operation-level parallel. The image-level parallel is a high-level parallel which divides one image into different parts and processes them concurrently. The operation-level parallel, which is embedded in each image-level parallel thread, fully explores every parallel part inside the concrete algorithms. The corresponding design is based on a DE2 Development Board which contains a CYCLONE II FPGA device. Meanwhile, the same task has also been implemented on PC and DSP for performance comparison. Despite the fact that operating frequencies of used PC and DSP are much higher than FPGA's, FPGA costs less time per computed image than both of them. By taking advantage of the double-parallel technique, the speed/frequency ratio of FPGA is 202 times faster than PC and 147 times faster than DSP. Finally, a detailed discussion about different platforms is conducted, which analyzes advantages and disadvantages of used computing platforms. This paper reveals that the proposed double-parallel scheme can dramatically speed up image processing methods even on a low-cost FPGA platform with low frequency and limited resources, which is very meaningful for practical applications.

Index Terms—Double-parallel, image filter, edge detection, FPGA.

I. INTRODUCTION

Image filter and edge detection are significant techniques in the field of image processing. Many algorithms used in image filter and edge detection are highly parallelizable, which can be sped up dramatically with a custom circuit like FPGA. Due to FPGA's superior flexibility, rich resources and powerful parallel computing capacity, implementing highly parallelizable algorithms on FPGA is meaningful. Some researchers have achieved hardware implementation of median filter methods on FPGA but without image-level parallel[1], [2], [3], [4], [5], which have not adequately exploited FPGA's parallel

computing capacity. Besides, edge detection algorithms have also been implemented on FPGA [6], [7], [8]. But these edge detection steps also lack the image-level parallel. Aforementioned problems make implementing efficient algorithms of image filter and edge detection on FPGA necessary.

In order to solve above-mentioned problems, a double-parallel image processing architecture based on FPGA is proposed in this paper for median filter and edge detection. The double-parallel architecture contains two levels of parallel: image-level parallel and operation-level parallel. The experimental results show that the two-level parallel structure is very suitable for accelerating highly parallelizable image processing tasks. The double-parallel image processing architecture provides an efficient method for meeting the real-time demand in many applications, such as intelligent vehicles, the robotic vision system[9], [10].

The reminder of this paper is organized as follows. Section II describes median filter and edge detection methods. Section III presents the implementation of the whole system and detailed discussions on each module. This is followed by Section IV, where experimental results and analysis on different platforms are discussed. The last section is a final conclusion as well as an outlook.

II. DESCRIPTIONS OF MEDIAN FILTER AND EDGE DETECTION METHODS

In this paper, improved median filter and Prewitt edge detection methods are applied for image smoothing and edge extraction. The Median filter is a common non-linear filter, which replaces one pixel with the median value of its neighborhoods[11]. Different neighborhoods defined by filter masks result into different median filter methods[12]. In this paper, the standard median filter algorithm is improved to meet the parallel architecture of FPGA. A 3×3 filter mask is used to process a 2-Dimensional (2D) traffic image. Instead

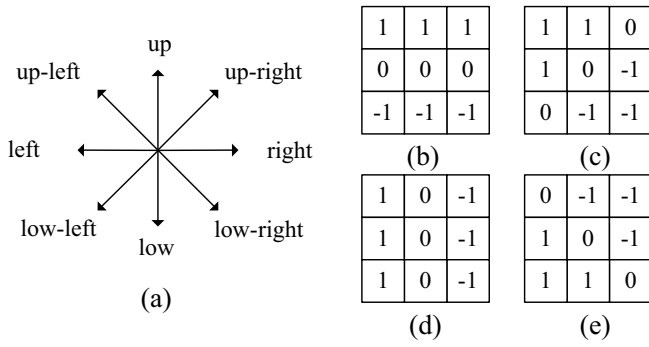


Fig. 1. Convolution kernels of the Prewitt edge detector. (a) Eight edge orientations. (b) Up orientation kernel. (c) Up-left orientation kernel. (d) Left orientation kernel. (e) Low-left orientation kernel.

of calculating the median value of 9 pixels around the center pixel, we extract the median value of 3 pixels in each row first. Then, the final median result is calculated from the three row output values. This improvement will greatly reduce the searching time of median values.

The Prewitt edge detection method is a classic algorithm for edge detection in image processing[13]. Through calculating the maximum response of a set of edge templates, it finds a direction of each pixel in which the intensity changes rapidly[14]. Various convolution kernels can be used as edge templates. When using a Prewitt edge detector in this paper, an input image is convolved with 8 3×3 convolution kernels, each of which is sensitive to a different orientation as shown in Fig.1 (a). Figure 1(b-e) respectively show up, up-left, left and low-left orientation kernels. For each pixel the output value is estimated by using the maximum response of these 8 edge templates at this pixel location.

III. SYSTEM IMPLEMENTATION

In this paper, the double-parallel architecture contains two levels of parallel: image-level parallel and operation-level parallel. The image-level parallel is a high-level parallel, which divides one image into different parts and processes them concurrently. The operation-level parallel is inside each thread of image-level parallel (high-level-thread), which fully explores every parallel part inside the concrete algorithms. The following subsections sequentially introduce the implementation of the whole system and main modules in the system.

A. System overview

A FPGA-based system-on-chip (SoC) usually contains several soft core processors[15]. In this case soft core processors control the work of the system. In this paper, a NIOS-II CPU [16] is used to control the whole double-parallel image processing system, which includes data transfer and image display. The structure of the double-parallel system is shown in Fig.2. The soft core processor, which is inside the big square frame, includes a NIOS-II CPU, two pairs of median filters and edge detectors, a synchronous dynamic random access memory (SDRAM) controller, a SRAM controller,

several direct memory accesses (DMAs), a video graphics array (VGA) controller and a timer. The soft core gets image data from an external memory by using a DMA as this system starts up. Then, imaging processing modules in Fig.2 respectively process different parts of input data in parallel, which are the image-level parallel. Finally, the soft core sends the result to an external display memory for displaying it on VGA.

The image-level parallel can be found in Fig.2, which divides an input image into two parts. The two parts are respectively transferred to their own image processing modules. Then, the two same image processing modules process different image parts, respectively. This parallel scheme is very effective in accelerating image processing. When using this parallel architecture, it is necessary for the whole system to solve the data synchronization problem between different threads. As shown in Fig.2, DMA1 delivers a half image into on-chip memory1 and the other half image into on-chip memory2. Then, the NIOS-II CPU starts DMA2, DMA3, DMA4, and DMA5 synchronically. DMA2 and DMA3 concurrently send image data to their respective image processing modules, while DMA4 and DMA5 receive results from two image processing modules. Finally, DMA1, which is used to deliver results to output image memory for displaying image, is started up until both DMA4 and DMA5 have finished data transmission.

The operation-level parallel is shown in Fig.3, which exists in both each median filter and each edge detector. During the median filter step 9 neighboring pixels including the center pixel are assigned to three row extractors for shortening the searching time of the median value. At first, each row extractor extracts the median value of three pixels in its row. The three row extractors work in parallel, which is one operation-level parallel. Then, the final median extractor calculates the median value of the output values of three row extractors. This improvement will greatly expedite the process of median searching. Although the final result is not the exact median of nine pixel values, it is among the middle three values and brings very little errors for an edge detection step. After a median filter step filtered pixels are sent to an edge detector. For each pixel gradients of eight directions are calculated in parallel, which is another operation-level parallel. Then, the maximum gradient is calculated from these eight gradients.

B. Hardware implementation of median filter module

The construction of hardware modules are based on MATLAB simulink tool and Altera corporation's DSP Builder library[17]. Firstly, hardware function modules are conducted by using a component library in simulink. Secondly, the signal compiler in DSP Builder library translates symbolic language into VHDL language. Finally, VHDL files are burned into FPGA. The hardware structure of the median filter is shown in Fig.4(a). There are nine pixel input ports, which transfer data synchronously by the use of delay components[17]. Meanwhile, there are four median extractors: three row extractors and one final extractor. Each median extractor inputs three

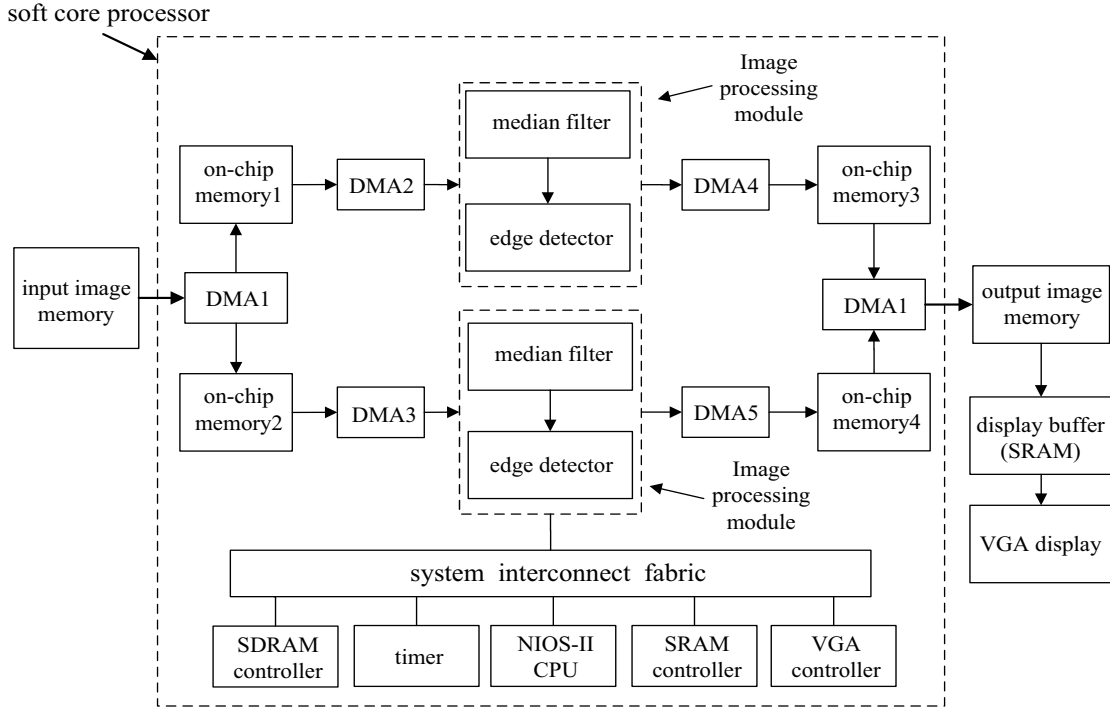


Fig. 2. Double-parallel image processing system.

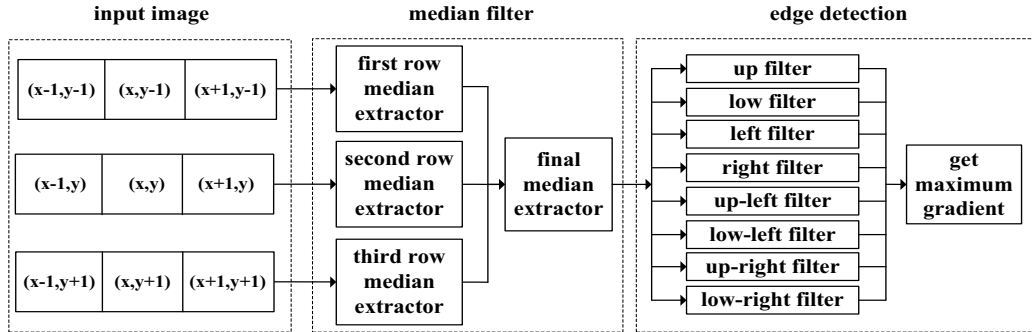


Fig. 3. Operation-parallel in each median filter and each edge detector.

pixel values and generates a median value. The structure of the median extractor is shown in Fig.4(b), which consists of comparator, multiplexer and if-statement components[17].

C. Hardware implementation of edge detection module

The hardware structure of the edge detection module is illustrated in Fig.5, where 8 orientation convolutions are packed to 8 gradient filters. As the space is limited, only two gradient filters (up filter and low-right filter) are shown. For each gradient filter, it inputs six involved pixels and generates a gradient of its orientation. Then, the maximum gradient value is calculated from these gradient values by four max modules. Each max module exploits comparator and multiplexer components to get the maximum value of three input gradient values. Finally, a comparator is used to

normalize the final output value to 0-255.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

A. Boundary processing of the divided image

During image-level parallel operation, the input image is divided into two sections for parallel median filter and edge detection. In this paper, both median filter and edge detector utilize 3×3 masks, which means that processing each pixel, as well as the pixels in the boundary of each section, needs eight pixels around it. In order to deal with the boundary problem, each section contains four more rows of pixels than half input image. The four extra rows are responsible for processing a boundary row.

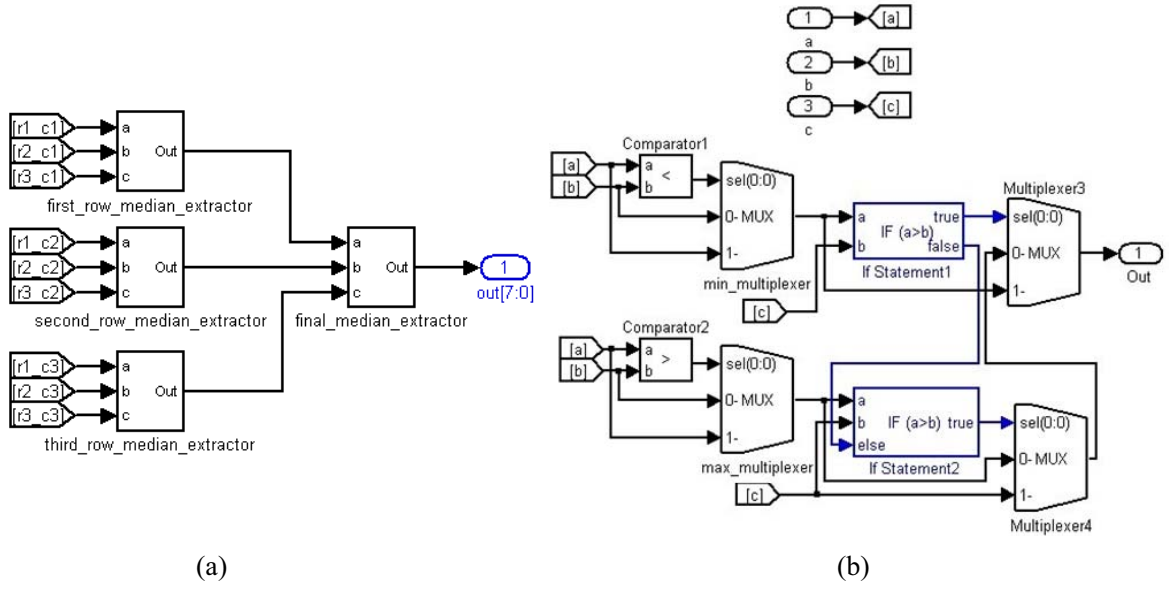


Fig. 4. View of the median filter module. (a) Hardware structure of the median filter module. (b) Hardware structure of the median extractor.

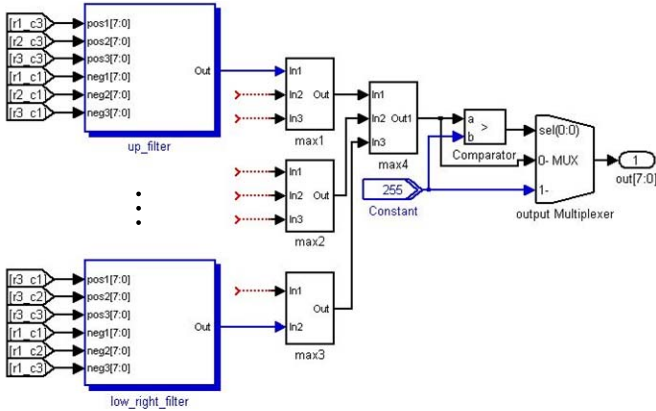


Fig. 5. Hardware structure of edge detection module.



Fig. 6. FPGA platform.

B. Experimental results and analysis on different platforms

Figure 6 shows our FPGA platform and the output traffic image on VGA.

For the purpose of testing the proposed double-parallel architecture, both median filter and edge detection methods have also been implemented on PC and DSP. Microsoft Visual Studio 2005 and CCS2.20.18 are respectively used as the program tool on PC and DSP, and C language is the program language on both the two platforms. On PC and DSP, the same median filter and edge detection algorithms are achieved by using single thread. The image data, which are memorized in an array, are convolved with a 3×3 median filter mask pixel by pixel. Then, the filtered image data are convolved with 8 3×3 edge templates pixel by pixel to calculate the gradients of 8 directions as shown in Fig.1. Finally, After we compare

the maximum of 8 gradients with 255, the smaller value is used as the output value.

The hardware information of three different platforms are illustrated in Tab. I. In comparison experiments, all three platforms perform the same filter and edge detection tasks on two traffic images with noise. The difference is that FPGA adopts the double-parallel architecture while PC and DSP use serial programs in a single thread. The two used traffic images differ in size, but not in content. The sizes of them are 80×60 and 160×120 with 8-bit unsigned char type. The original image with 160×120 size and three results of different platforms are shown in Fig.7.

We can see that three results are nearly the same, which

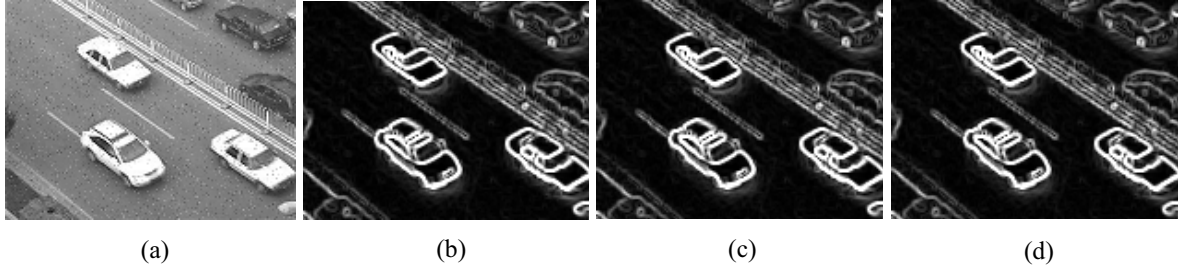


Fig. 7. Comparison experiments. (a) Original 160×120 traffic image. (b) Result of FPGA platform. (c) Result of PC platform. (d) Result of DSP platform.

TABLE I
HARDWARE INFORMATION OF THREE EXPERIMENTAL PLATFORMS.

Platform	Device	System clock
FPGA	Cyclone II EP2C35F672C6	50MHZ
PC	Intel core™ 2 Quad CPU	2.66GHZ
DSP	TMS320DM642	600MHZ

TABLE II
TIME-CONSUMPTION OF THREE PLATFORMS.

Image size (pixel)	80×60	160×120
Time of PC without multiplication (s)	0.00032650	0.00150938
Time of PC with multiplication (s)	0.00061757	0.00250071
Time of FPGA without multiplication (s)	0.00019936	0.00065764
Time of FPGA with multiplication (s)	0.00020030	0.00065860
Time of DSP without multiplication (s)	0.00196213	0.00871805
Time of DSP with multiplication (s)	0.00184613	0.00811296

TABLE III
TIME COST RATIO OF PC, FPGA AND DSP ON 160×120 TRAFFIC IMAGE.

Ratio	PC/FPGA	DSP/FPGA
Frequency ratio	$\frac{2.66GHZ}{50MHZ} \approx 53$	$\frac{600MHZ}{50MHZ} = 12$
Time ratio without multiplication	$\frac{0.00150938}{0.00065764} \approx 2.30$	$\frac{0.00871805}{0.00065764} \approx 13.26$
Time ratio with multiplication	$\frac{0.00250071}{0.00065860} \approx 3.80$	$\frac{0.00811296}{0.00065860} \approx 12.32$

commendably demonstrates the validity of our implementation. Moreover, the running time of all three platforms on test images are recorded. When calculating eight gradients in the edge detection step, multiplication operations are needed between pixel values and convolution kernels. There are at least 9 multiplication operations per gradient calculation and 72 per 8 gradients calculation, which are heavy burdens to computing platforms. So an improved method is used when calculating gradient values, which unfolds the convolution operation and changes multiplication to addition and subtraction. Both running time with and without multiplication are recorded in Tab. II. It is obvious from the sheet that FPGA is faster than used PC and DSP. By taking advantage of the double-parallel architecture, FPGA is 2.29 times faster than PC and 13.26 times faster than DSP on 160×120 image without multiplication operation.

For a detailed speed comparison of the three platforms, time cost ratios of these platforms on 160×120 image are listed in Tab. III. Although the PC's clock is 54 times faster than FPGA, it has a poorer performance. It also can be found that time cost ratio of PC/FPGA changes from 2.30 to 3.80 when using multiplication operations, while DSP/FPGA changes from 13.26 to 12.32. The cause of the PC's performance declining much more than FPGA is that FPGA adopts embedded hardware multipliers and all multiplications in FPGA are in parallel. Relatively speaking, PC has no embedded hardware multiplier, which greatly slows the processing speed when a large quantity of multiplication operations are required. Compared with PC, DSP has hardware multipliers, so performances of DSP with multiplication is nearly the same as that without multiplication. Meanwhile, the speed/frequency ratios of the three platforms are also calculated for an objective evaluation of performances. By taking advantage of the double-parallel technique, the speed/frequency ratio of FPGA is 202 times faster than PC and 147 times faster than DSP on 160×120 image with multiplication.

V. CONCLUSION

A double-parallel architecture based on FPGA for median filter and edge detection is proposed in this paper. The experimental results show the high performance of the proposed architecture. Meanwhile, hardware implementation based on FPGA and analysis between different platforms are presented

in detail, which will be meaningful for other engineers.

As resources of FPGA used in this paper are limited, the image-level parallel in this paper only has two threads, which decreases the performance of the proposed architecture to a certain degree. If a more powerful FPGA is applied, FPGA can employ more threads in parallel and behaves better. In next step, we will do the experiment about a trial of dividing input image into more parts than two to accelerate the image processing speed. Moreover, the double-parallel architecture can also be applied to other image processing methods.

VI. ACKNOWLEDGEMENT

We would like to express our sincere appreciation to professor Fei-Yue Wang for his instruction and encouragement. This work is supported in part by NSFC 70890084, 60921061, 90920305; CAS 2F09N05, 2F09N06, 2F10E08, 2F10E10.

REFERENCES

- [1] P. Wei, L. Zhang, C. Ma, and T. S. Yeo, "Fast median filtering algorithm based on fpga," in *Signal Processing (ICSP), 2010 IEEE 10th International Conference on*, 2010, pp. 426–429.
- [2] C. J. Juan, "Modified 2d median filter for impulse noise suppression in a real-time system," *Consumer Electronics, IEEE Transactions on*, vol. 41, no. 1, pp. 73–80, Feb. 1995.
- [3] R. Maheshwari, S. Rao, and P. Poonacha, "Fpga implementation of median filter," in *VLSI Design, 1997. Proceedings., Tenth International Conference on*, Jan. 1997, pp. 523–524.
- [4] Y. Hu and H. Ji, "Research on image median filtering algorithm and its fpga implementation," in *Intelligent Systems, 2009. GCIS '09. WRI Global Congress on*, vol. 3, May 2009, pp. 226–230.
- [5] Y. Lu, M. Dai, L. Jiang, and S. Li, "Sort optimization algorithm of median filtering based on fpga," in *Machine Vision and Human-Machine Interface (MVHI), 2010 International Conference on*, 2010, pp. 250–253.
- [6] Z. Shanshan and W. Xiaohong, "Vehicle image edge detection algorithm hardware implementation on fpga," in *Computer Application and System Modeling (ICCASM), 2010 International Conference on*, vol. 14, 2010, pp. 184–188.
- [7] K. El Houari, B. Cherrad, and I. Zohir, "A software-hardware mixed design for the fpga implementation of the real-time edge detection," in *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, 2010, pp. 4091–4095.
- [8] Z. Guo, W. Xu, and Z. Chai, "Image edge detection based on fpga," in *Distributed Computing and Applications to Business Engineering and Science (DCABES), 2010 Ninth International Symposium on*, 2010, pp. 169–171.
- [9] L. Li, J. Song, F.-Y. Wang, W. Niehsen, and N.-N. Zheng, "Ivs 05: new developments and research trends for intelligent vehicles," *Intelligent Systems, IEEE*, vol. 20, no. 4, pp. 10–14, july-aug. 2005.
- [10] F.-Y. Wang, P. J. Lever, and B. Pu, "A robotic vision system for object identification and manipulation using synergetic pattern recognition," *Robotics and Computer-Integrated Manufacturing*, vol. 10, no. 6, pp. 445–459, 1993.
- [11] L. Yin, R. Yang, M. Gabbouj, and Y. Neuvo, "Weighted median filters: a tutorial," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 43, no. 3, pp. 157–192, Mar. 1996.
- [12] T. Nodes and J. Gallagher, N., "Median filters: Some modifications and their properties," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 30, no. 5, pp. 739–746, Oct. 1982.
- [13] T. Peli and D. Malah, "A study of edge detection algorithms," *Computer Graphics and Image Processing*, vol. 20, no. 1, pp. 1–21, 1982. [Online]. Available: <http://www.sciencedirect.com/science/article/B7GXF-4D7JP09-27/2/f4a8a3ff110b8635fea9eeca0259b305>
- [14] Z. Guo, W. Najjar, F. Vahid, and K. Vissers, "A quantitative analysis of the speedup factors of fpgas over processors," in *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, ser. FPGA '04. New York, NY, USA: ACM, 2004, pp. 162–170. [Online]. Available: <http://doi.acm.org/10.1145/968280.968304>
- [15] Altera Corporation, "Creating multiprocessor nios ii systems tutorial." February, 2010.
- [16] Nios-II Integrated Development Environment. Available: <http://www.altera.com/literature/lit-index.html>.
- [17] Altera Corporation. June, 2010, DSP Builder Handbook Volume 2: DSP Builder Standard.