International Journal of Multimedia and Ubiquitous Engineering
Vol.9, No.8 (2014), pp.367-376
http://dx.doi.org/10.14257/ijmue.2014.9.8.34

A Real-time Shape Recognition Scheme for Hardware Implementation

Young Chun Kwon[1,2], Yeon-woo Kim[1], Sunhan Jeong[1] and Nakhoon Baek[2]

[1]*Samsung Electronics, Suwon, Korea*
[2] *School of Computer Sci. and Eng., Kyungpook National Univ., Daegu 702-701, Korea*
kown100@nate.com, sanaeida@nate.com, sunahn86@nate.com, oceancru@gmail.com

Abstract

We present a shape recognition scheme for real-time embedded systems. It was hard to achieve real-time processing on the typical embedded systems. After analyzing the overall image processing and shape recognition process, we introduced massively parallel processing schemes, and finally implemented special purpose FPGA chips. Strengthen with this FPGA implementation, our target board achieved real-time shape recognition. A two-leg walking robot was directed to the target positions after the shape recognition, for demonstration purpose. We will extend these real-time shape recognition schemes for other image processing applications.

Keywords: *shape recognition scheme; real-time processing; robot motion*

1. Introduction

When processing a digital image with pixel-wise masking operations, most of its processing time is used for repeatedly applying the same operation to each pixel [1]. These kinds of repeated operations are one of the most suitable ones to be accelerated by parallel processing techniques. With current parallel computer hardware, we have plenty of parallel-processing methods for these image processing operations.

There are lots of hardware devices based on these kinds of simple digital image processing techniques [2]. These devices generally have various hardware specifications, while their purposes are usually restricted to the pre-defined areas. Thus, it is most important to find the most optimal way of achieving the required functionality with restricted resources.

In this paper, we aim to apply shape recognition and image-processing techniques for the real-time applications. A two-leg walking robot was used for the demonstration of our final implementation. As its main computing power, this robot is equipped with an embedded board, which is actually hard to achieve image processing operations in real-time. Thus, we accelerated the given hardware with parallelized special hardware: an FPGA chip [3].

We analyzed the required image processing operations, and designed overall architecture. To design our special-purpose FPGA chips, we used HDL (hardware description language) [4]. Our FPGA implementation of parallel image processing algorithms enables us to achieve the real-time shape recognition of the walking robot, whose motions are finally controlled in real-time to grab and remove those target objects.

* Corresponding author: Nakhoon Baek, oceancru@gmail.com

† This investigation was financially supported by Semiconductor Industry Collaborative Project between Kyungpook National University and Samsung Electronics Co. Ltd.

ISSN: 1975-0080 IJMUE

Our implementation successfully shows the way of accelerating low-tier embedded systems with FPGA-based parallel processing.

We will start from the previous works and related results in Section 2. Section 3 represents our design of the parallelized shape recognition scheme, and also our FPGA implementation. Implementation results are in Section 4, and finally conclusions are followed in Section 5.

2. Previous Works

In this section, we will show some examples to accelerate image processing algorithms with various methods including SIMD architectures [5], OpenMP [6], and others. In these days, most CPU's support SIMD (single instruction multiple data streams) instructions. A quad-core CPU [7] is used for our experiments. Basically, SIMD architecture requires parallel processing. In our implementation, we have implemented totally five image processing algorithms of median filter, mean filter, morphology operation, edge detection and sharpening with SIMD parallel processing instructions.

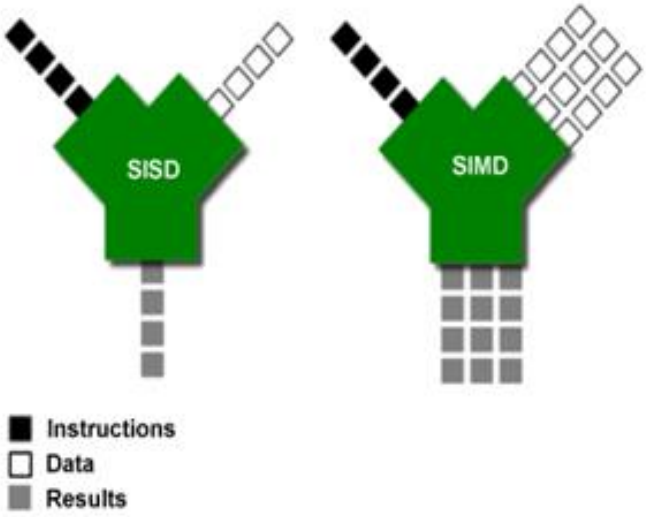


Figure 1. Characteristics of SISD and SIMD Processing

Figure 1 shows the characteristic differences between SISD (single instruction single data stream) and SIMD. In SIMD architecture, the data is stored into a single array structure, and the SIMD architecture processes multiple units of data for each instruction. For example, a chunk of 128bit data can be composed of four 32bit data units, as shown in Figure 2. Then, a single SIMD instruction will process all these four 32bit data in a single step. To parallelize the target image processing algorithms, we need those parallelized mathematical instructions provided by the standard SIMD library.

Another parallel processing library of OpenMP [6] provides more parallelized high-level processing based on the multiple cores in the current CPU architecture. In typical image processing applications, they frequently use double for-loop to process 2D pixel information. OpenMP provides parallel processing features for these for-loop's and other sequential processing, according to the number of available CPU cores. Figure 3 shows a typical OpenMP processing architecture. The master thread automatically generates its children threads according to the number of CPU cores and multiprocessing requirements. In its design, it is important to isolate the parallelized portions and the sequential portions of the whole process, to maximize the OpenMP capabilities.

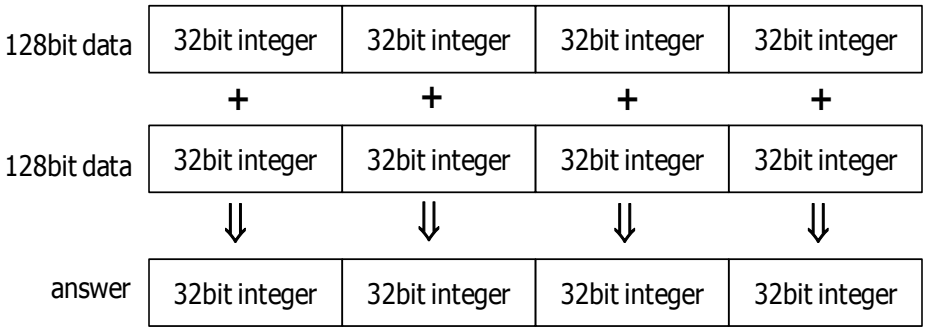


Figure 2. An Example of SIMD Operations with 128bit Data Chunk

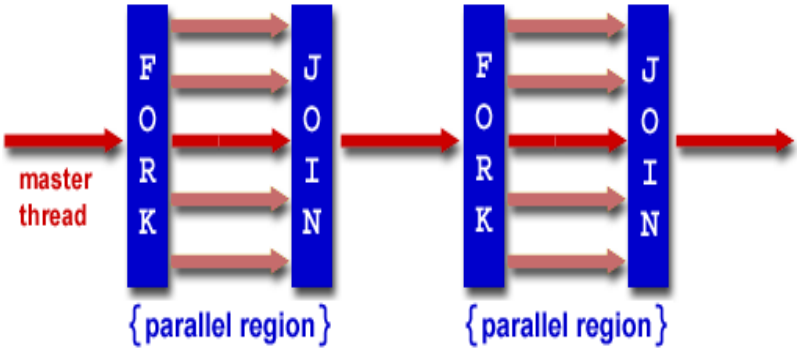


Figure 3. A typical OpenMP Processing Architecture

There are also some hybrid approaches, which use both of SIMD instructions and OpenMP processing, to maximize the overall performance. In the field of image processing applications, it is also common to use massively parallel processing techniques with GPGPU (general purpose graphics processing unit).

In this paper, we implemented a special purpose image processing chip, and install this FPGA chip onto the existing image processing device. We will show how much we can accelerate the existing device with our custom FPGA parallelized processing chip, with detailed experiment results.

3. Design and Implementation

The target embedded device is an ARM-based development board, with uCLinux Operating System. More specifically, it is equipped with the Eagle-series main CPU, 64MB SDRAM, 64MB NAND Flash memory, and an FPGA connection with 16bit address and data buses [8].

Our overall system architecture is represented in Figure 4. The simulator part has actually three different versions of image processing algorithms: a PC-based software implementation, another target embedded device-based software implementation, and our FPGA-based parallelized implementation. We can execute a specific image processing algorithm for all three implementation, and compare the results each other. The two-leg walking robot acts as an I/O device. Its internal camera gets the digital images and transfers those images to the embedded board. The robot motion control algorithm works on this embedded board, with the computing support of the FPGA chip. Finally, our robot detects some geometric objects as obstacles in its walking path.

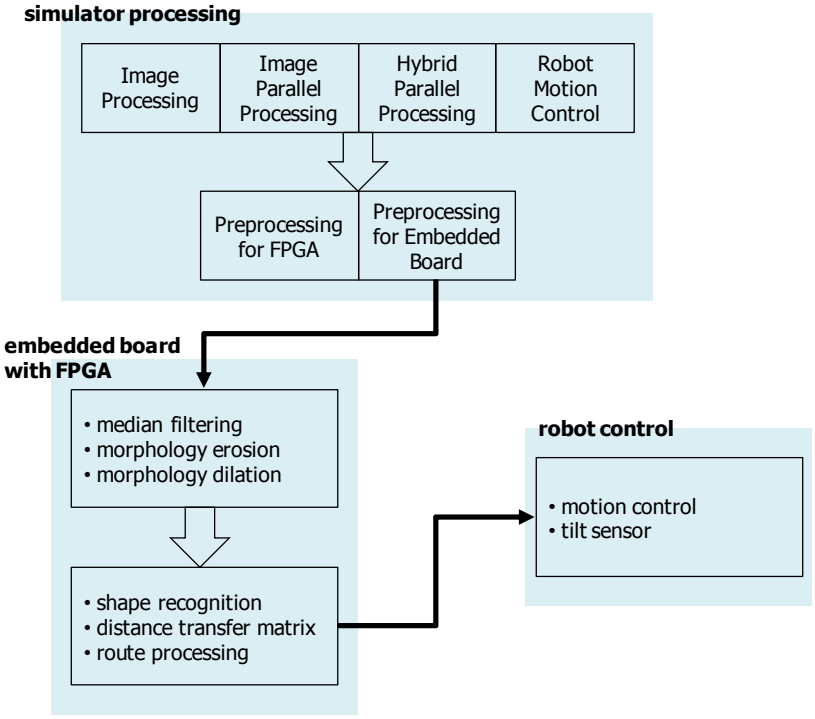


Figure 4. Overall Architecture of our System

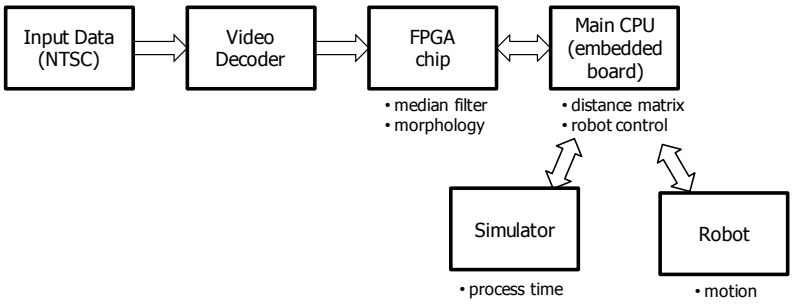


Figure 5. Information Flow for the Embedded Board and our FPGA Chip

The camera-captured images from the robot are 320×240 resolution with RGB colors. At the first stage of our algorithm, we apply a median filter to remove potential noises. And then, a morphology filter is used to additionally remove noises in the image.

The two-leg walking robot should recognize the target shapes with pre-specified colors, to trace the target object, against the real-time environmental changes. Figure 5 shows our overall processing to continuously trace the target shape from the input images. These features are simultaneously implemented for both of the FPGA implementation and the parallel processing simulator on the target embedded board. Our implementation focuses on the parallel processing, and we will compare the result with and without our FPGA implementation.

At the first stage, it gets the input RGB image, with 320×240 resolutions. Then, potential noises in the input images are removed with the median filters. To remove more noises and improve the recognition success ratio, we apply additional morphology operations. At the next stage, the distance-transformation matrices are used to isolate the candidate regions for the geometric objects [9]. It also checks their RGB colors with color correction techniques. Additional location information is used to identify those geometric objects in the input images.

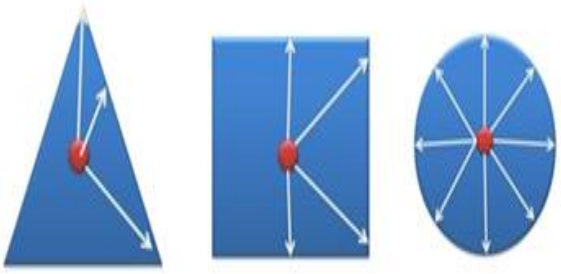


Figure 6. Shape Recognition from the Center Point

Based on the distance-transformation matrices, the target regions are isolated, and then, the RGB values for those regions are checked with the pre-specified shape and colors. For the shape recognition, as shown in Figure 6, we first get the center point from the distance-transformation matrix, and enlarging the target region to finally isolate the final shape region. After recognizing the shape, the color values, and the center point, we finally indicate the target object on the screen.

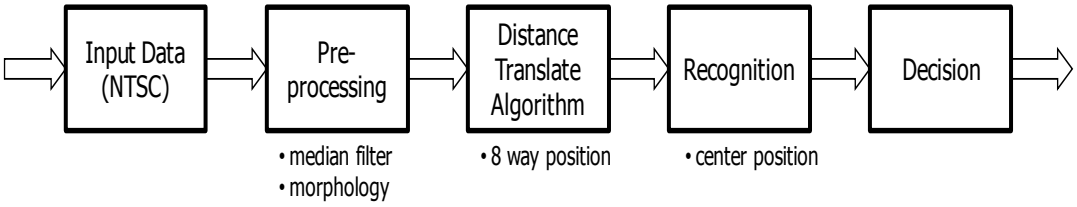


Figure 7. Color Image Conversion and Recognition Process

From the NTSC-type video camera, the images are captured and converted to digital images. Then, our FPGA chip performs a median filter and another morphology filter on those images. The main CPU processes the distance-transformation matrix algorithm in its sequentially implemented execution. Then, the motion control algorithms for the walking robot are executed to make required motions of the robot.

We use an embedded board with Eagle CPU, which is a variation of ARM processors. The uCLinux operating system [10] is installed on that board. The embedded board has 64MB SDRAM and 64MB NAND Flash Memory, connected to the FPGA chip with 16bit address bus and data bus. Figure 7 shows the data flow on the embedded board. The analog signals from NTSC-type camera are converted to digital signals. These digital signals are processed with our median filter algorithm and morphology processing, implemented on the FPAG chip. Then, the distance-transformation matrix algorithm is applied to recognize the target shapes. All the related information is then delivered to the robot, with the appropriate actions for the robot. Our extra simulator also processes the same processing steps, to finally get the comparison results.

In our hardware configuration, input images are first obtained in the YUV color format. The YUV color format is used to express the analog intensity and color values in the component video system. The luminance component in the Y channel expresses the black and white signal. The UV representation of chrominance was chosen over straight R and B signals because U and V are color difference signals. Thus, YUV signals are typically created from RGB (red, green and blue) source. Weighted values of R, G, and B are summed to produce Y, a measure of overall brightness or luminance. U and V are computed as scaled differences between Y and the B and R values.

Finally, the RGB values can be obtained from the YUV signals as follows:

$$R = 1.164\ (Y - 16) + 1.596\ (V - 128)$$

$$G = 1.164\ (Y - 16) - 0.813\ (V - 128) - 0.391\ (U - 128)$$

$$B = 1.164\ (Y - 16) + 2.018\ (U - 128)$$

In our image processing chip, we have implemented edge detection methods in addition to the median filtering. On the uCLinux system and the FPGA chip, we applied various image processing algorithms on the pre-processing steps from the robot-captured images.

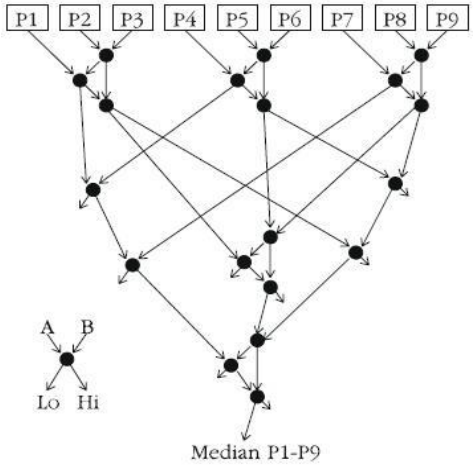


Figure 8. Our Combinational Logic Design

To achieve a real-time median filtering implementation, it is better to use a combinational logic, based on the parameter comparison method, as shown in Figure 8. For FPGA implementations, we use a Verilog-based design of the median filter, shown in Figure 9. Now, this median filter implementation picks up the median value from a pixel and its 8 surrounding pixels.

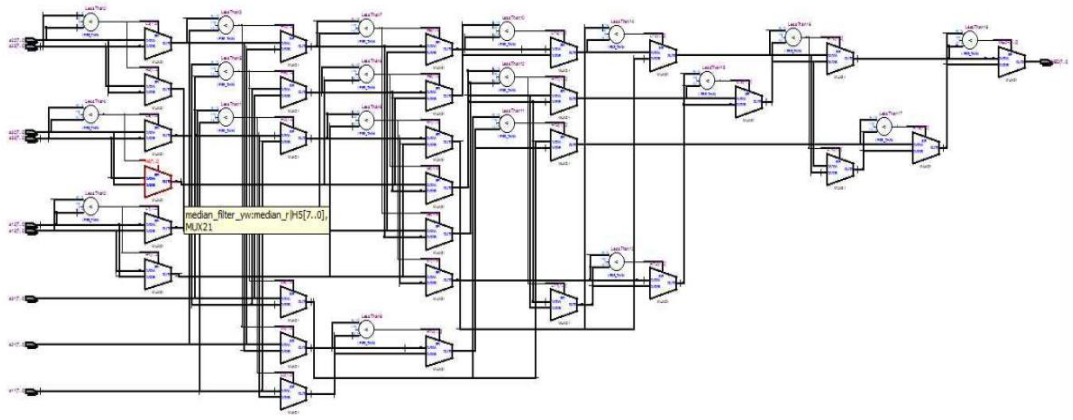


Figure 9. Our FPGA Design

Morphology erosion and dilation operations require some masking operations. Thus, our implementation uses type-D flip-flops, as shown in Figure 10. Using counter logics, our implementation can achieve successive morphology operations. Additional AND and OR gates are used to achieve the final morphology circuit.

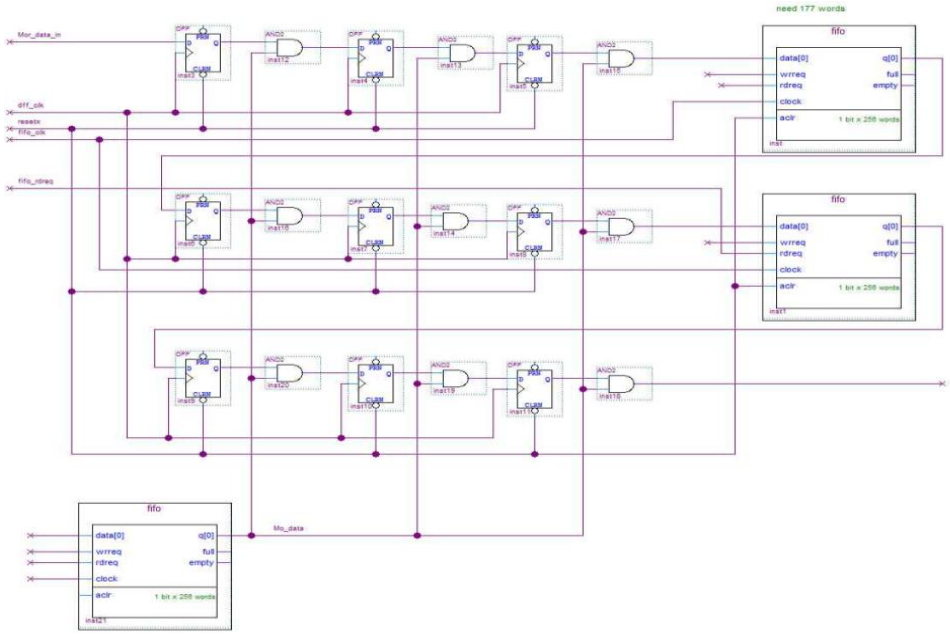


Figure 10. Our Counter Logic Design

4. Implementation Results

Figure 11 shows the user interface of our robot simulation program. It consists of the parallel processing window, the mode selection window, the image processing algorithm window, the two-leg robot control unit, the serial communication unit, the RGB image processing unit, the instruction unit, and the embedded board control unit.

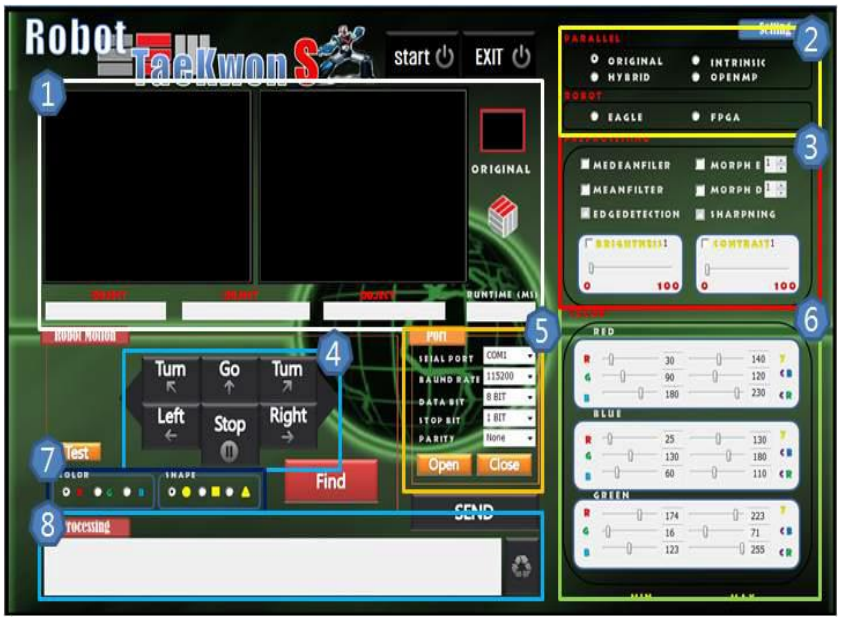


Figure 11. Our Robot Simulation Program

We have implemented totally 5 image processing steps, from median filtering to edge detection, as shown in Table 1. For the uCLinux and FPGA implementations, we choose some of them to achieve the best performance with limited resources. As shown in Table 1, we achieved at least 3 times speed up, through converting the original C source code to

parallelized implementations with OpenMP and SIMD instructions. Notice that we cannot achieve 4 times speed up exactly, even we use the full-time performance of quad-core CPU, mainly due to the memory bus competitions.

Adding SIMD instructions, we can achieve more speed ups of at most 5 times and at least 2.5 times. Another consideration for the embedded board would be the down-graded speed of the CPU. Due to the low-powered CPU's, our target embedded board is about 25 times slower in comparison with the ordinary PC's. However, combining the target embedded board and our FPGA implementation, we can achieve faster or equivalent speed in comparison to the ordinary PC's.

To make FPGA implementations, we carefully selected a set of mask-based image processing algorithms: median filter, morphology filter, mean filter, sharpening filter, and edge detection filter. For comparison purpose, we actually implemented all these algorithms on three different forms: a PC-based software implementation, an embedded board-based software implementation, and an FPGA implementation.

In the case of the embedded board-based software implementation, the target embedded board has relatively low computing power, and thus, shows much slow execution speeds. For example, the median filter algorithm works in 1,070 msec on the target embedded board, while the PC-version works in 44 msec, which is about 25 times faster. Our FPGA implementation fully utilizes the speed-up of parallelized implementation, and shows much faster execution speeds of 97 msec, as shown in Table 1.

Table 1. Execution Speed for Each Steps, (unit: msec)

	Median Filter	Morphology	Mean Filter	Sharpening	Edge Detection
original	44	56	46	50	77
OpenMP	13	21	15	18	24
SIMD	19	30	20	25	33
Hybrid	8	16	13	16	20
uCLinux	1073	1305	N/A	N/A	N/A
FPGA	97	102	N/A	N/A	N/A

For the practical experiments, we have checked the success and failure of the two-leg walking robot, as shown in Table 2. We provide three different types of target objects: circles, triangles, and squares. We tested 30 times for each object, to both of embedded-board based software implementation and our FPGA-based parallelized implementation. As shown in Table 1, our FPGA implementation outperforms the software implementation for circles and triangles, while shows similar score for squares. Conclusively, our FPGA-based parallelized implementation outperforms the software-based implementation, with remarkable speed-ups.

Table 2. Experimental Results for Geometric Objects

	target geometric object		
	circles	triangles	squares
Software implementation (on embedded boards)	25 success / 30 trials	24 success / 30 trials	27 success / 30 trials
our FPGA-based parallelized implementation (FPGA chips)	28 success / 30 trials	25 success / 30 trials	26 success / 30 trials

5. Conclusion

In this paper, we realized specific image processing algorithms into their parallelized implementations, as FPGA chips, for relatively low-powered embedded devices. Our final analysis shows that these FPGA implementations mark final scores equal to or better than typical embedded device implementations. In contrast, from the processing time point of view, it works much faster than embedded device implementations, and even similar

speed to the PC implementations. Our implementation results show that parallelized implementations of image processing algorithms on FPGA chips can supplement the low computing power of typical embedded devices.

Acknowledgements

This investigation was financially supported by Semiconductor Industry Collaborative Project between Kyungpook National University and Samsung Electronics Co. Ltd.

References

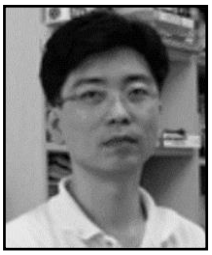
[1] M. Petro and C. Petrou, "Image Processing: The Fundamentals", Wiley, (**2010**).
[2] G. Blake, R. Deslinski and T. Mudge, "A survey of multicore processors: A review of their common attributes", IEEE Signal Processing Magazine, vol. 26, no. 6, (**2009**), pp. 26–37.
[3] A. Basheer and M. Zaghlool, "FPGA-Based High Performance Parallel Computing", Scholar's Press, (**2014**).
[4] D. Thomas and P. Moorby, "The Verilog Hardware Description Language", Springer, (**2008**).
[5] D. Patterson and J. Hennessy, "Computer Organization and Design", Fifth Ed.: The hardware/software interface, Morgan Kaufmann, (**2013**).
[6] R. Cook, "An Introduction to Parallel Programming with OpenMP, PThreads and MPI", Amazon Digital Service, (**2011**).
[7] Intel: Intel Core 2 Quad Processors, Intel, (**2013**).
[8] ADC, Inc, Eagle board Architecture diagram, application: http://www.adc.co.kr/product/
[9] G. Grevera, "Distance Transform Algorithms and Their Implementation and Evaluation, Deformable Models", Springer (**2007**), pp. 33-60.
[10] uclinux, Embedded Linux/Microontroller Project, (**2013**).

Authors

Young Chun Kwon, he was a master student in School of Computer Science and Engineering, Kyungpook National University. He is now a researcher in Samsung Electronics, Inc. His interests include computer graphics and parallel process.

Yeon-woo Kim, he is now a researcher in Samsung electronics. His interests include embedded system applications.

Sunhan Jeong is now a researcher in Samsung electronics. His interests include FPGA.



Nakhoon Baek, he is currently an associate professor in the School of Computer Science and Engineering at Kyungpook National University, Korea. He received his B.A., M.S., and Ph.D. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) in 1990, 1992, and 1997, respectively. His research interests include graphics standards, graphics algorithms and real-time rendering. He is now also the Chief Engineer of Mobile Graphics Inc., Korea.