Journal of Computers Vol. 29 No. 1, 2018, pp. 143-151
doi:10.3966/199115992018012901012

An Efficient VLSI Architecture of Direct Binary Search for Halftoning Image

Jihua Tang[1], Chih-Hsien Hsia[2*]

[1] School of Information Engineering, Longyan University, China

[2] Computer Science and Information Engineering, National Ilan University, Taiwan
chhsia625@gmail.com

Received 15 September 2017; Revised 29 November 2017; Accepted 10 January 2018

Abstract. The primary display technology utilized by today's printers is halftoning image, and among the various image halftoning methods that are currently being used, Direct Binary Search (DBS) is the method that produces the best image quality. However, one disadvantage of this method is the requirement that images be constantly iterated, as this places many limitations on the method with respect to processing speed. In view of this issue, this work proposes an effective hardware architecture design for the DBS method that would allow for image quality to be maintained and for real-time processing to be carried out. The study proposes the application of the line buffer method to the Very Large Scale Integration (VLSI) process, so as to reduce memory usage, enable the use of a parallel processing architecture. Thus, increase the speed of the process. As results, the DBS halftoning image method was used on a gray image, and it was found that, with respect to the image, the number of iterations used and the halftone image values generated by the hardware that was updated based on the visual model error table were consistent with the results generated by software algorithms.

Keywords: direct binary search, halftoning image, parallel processing architecture, very large scale integration

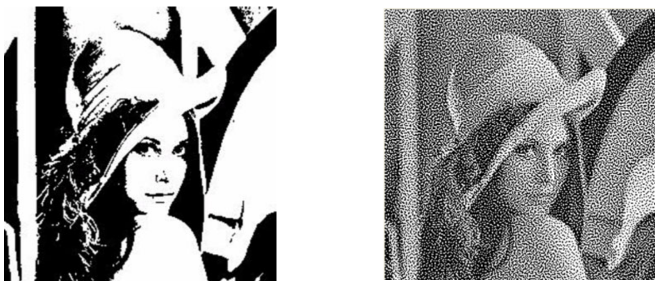
1 Introduction

A three-dimensional (3D) video sequence is composed of two or more videos captured by diverse view cameras Image digitization has led to constant advancements in multimedia storage and transmission. The display content of digital images is formed by millions of pixels, and the most common types of images used are gray and color images. Gray images consist only of data distinguished by varying levels of brightness and darkness, with the data being able to be quantified into 256 different continuous shades of gray, which means that 8 bits are required for a single pixel. Color images are formed from the colors red, green, and blue (RGB), with each color being able to be quantified into 256 shades, which means that 24 bits are required for a single color pixel. The multi-tone approach is currently used to render images because displays are now capable of displaying color depth. However, due to previous limitations imposed by hardware output devices and the characteristics associated with the display of a continuous color image, research into image halftoning techniques was conducted to explore ways to optimize the use of low-level hardware and equipment [1]. In halftone images, every pixel can either be black (0) or white (1) and thus requires only 1 bit for storage. In contrast, a binary image is binarized using threshold values, as shown in Fig. 1(a). In image halftoning, the difference in dot density is used to create gray images, and the dot density of a continuous gray image is higher when its tone is darker, and lower when its tone is lighter, and as a result, halftone images can lead to a significant reduction in data storage and transmission requirements, and express the characteristics of continuous gray images, as shown in Fig. 1(b).

Among the various image halftoning methods that are currently being used, Direct Binary Search,

[*] Corresponding Author

called DBS, is the method that produces the best image quality. However, its algorithm requires an inputted image to be constantly iterated, which makes the method the slowest in terms of computing speed [2]. Previous work on the DBS method has mostly focused on raising image quality, and no studies have been conducted on VLSI hardware implementation with the aim of increasing the method's computing speed. In view of this issue, this study aimed to utilize an integrated circuit design for the DBS method, so as to allow for image quality to be maintained and to resolve the computing speed issue. The DBS method was first proposed by Allebach et al. in a 1992 study [1] that focused on examining DBS generated halftone images as target images or comparative targets. Despite the study's high research value, its commercial value was limited. Therefore, this work proposes a hardware approach to reduce internal memory usage (which allows for hardware sizes to be reduced) and increase computing speed, and thereby raise the applicability of the DBS method in the market.



(a) Binarized image (b) Halftone image

Fig. 1. Binary image representation

The rest of this paper is organized as follows. Section 2 introduces the human visual system model and DBS method. Section 3 discussing software and hardware implementation and providing a description of the proposed VLSI architecture, the Section 4 detailing the study's hardware design simulation and verification,. Finally, Section 5 concludes this innovative method.

2 Direct Binary Search

2.1 Traditional Direct Binary Search Methods

The Human Visual System (HVS) model plays an important role as the generated halftone images have to emulate the continuity of gray images [3]. The human eye is more sensitive to low-frequency images and less likely to detect high-frequency ones, acting like a low-pass filter with a cut-off frequency that decreases when the observation distance increases [7]. By taking advantage of these properties, a model suitable for the human eye can be designed, and the HVS model can serve as objective basis for judging image quality [4].

A gray image is defined as $f[m,n]$ while a halftone image is defined as $g[m,n]$. The original halftone image $g[m,n]$ can be a random noise binary image and the HVS model can be defined as $\tilde{p}(x,y)$, as shown in Equs. (1)-(2). The objective of the DBS method is to select two images that look as close as possible to each other to the human eye, i.e. $\tilde{f}(x,y) \approx \tilde{g}(x,y)$. The error detected by the human eye is expressed in Equ. (3). Traditional DBS methods define E as the sum of squared differences and, as shown in Equ. (4), refer to it as the cost function.

As shown in Fig. 2, every pixel of a halftone image that is being iterated will be toggled and swapped, and therefore, 10 states (including the original state) must be considered for every pixel. As shown in Equ. (5), the cost functions are toggled or swapped, in the hope that the smallest cost function can be selected and updated, thus allowing the updated halftone image to more closely resemble the grayscale image. An iteration is complete after every pixel of an image has been considered, and the iteration process is repeated until the cost functions no longer decrease.

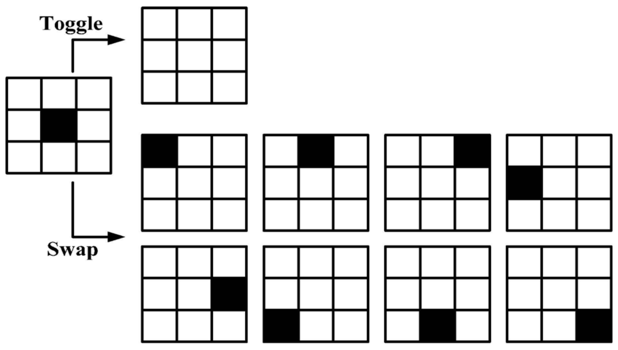


Fig. 2. Diagram showing toggling and swapping

$$\tilde{f}(x,y)=\sum_{m,n} f[m,n]\tilde{p}(x-mX,y-nY) \quad \textbf{(1)}$$

$$\tilde{g}(x,y)=\sum_{m,n} g[m,n]\tilde{p}(x-mX,y-nY) \quad \textbf{(2)}$$

$$\tilde{e}(x,y)=\tilde{g}(x,y)-\tilde{f}(x,y) \quad \textbf{(3)}$$

$$E=\left\|\tilde{g}(x,y)-\tilde{f}(x,y)\right\|^2 \quad \textbf{(4)}$$

$$E'=\left\|\tilde{f}(x,y)-\tilde{g}'(x,y)\right\|^2 \quad \textbf{(5)}$$

2.2 Quick Direct Binary Search

Some works [5-6] have proposed a simplified comparison method, in which the error between $\tilde{f}(x,y)$ and $\tilde{g}(x,y)$ can be redefined, as shown in Equ. (6). The error between the original image and the halftone image is defined as *e*[*m*,*n*], as shown in Equ. (7). The visual model error table is defined as $c_{\tilde{e}\tilde{p}}$ [*m*,*n*]. This table serves as the autocorrelation of the HVS model $\tilde{p}(x,y)$ that masks the results obtained after $c_{\tilde{p}\tilde{p}}$ [*m*,*n*] and *e*[*m*,*n*] are computed, as shown in Equ. (8).

$$\varepsilon=\sum_{m,n} e[m,n]c_{\tilde{e}\tilde{p}}[m,n] \quad \textbf{(6)}$$

$$e[m,n]=g[m,n]-f[m,n] \quad \textbf{(7)}$$

$$c_{\tilde{e}\tilde{p}}[m,n]=e[m,n]**c_{\tilde{p}\tilde{p}}[m,n] \quad \textbf{(8)}$$

The utilization of the output device's output discrete points by the autocorrelation function of the HVS model $\tilde{p}(x,y)$ is defined as $c_{\tilde{p}\tilde{p}}$ [*m*,*n*]. $c_{\tilde{p}\tilde{p}}$ [*m*,*n*] is treated as the mask for HVS model. In Equ. (9), the original $c_{\tilde{p}\tilde{p}}$ (*x*,*y*), and the $c_{\tilde{p}\tilde{p}}$ [*m*,*n*] that is derived is shown in Equ. (10), in which *R* and *D* are the output device resolution and human eye observation distance, respectively.

$$c_{\tilde{p}\tilde{p}}(x,y)=\iint \tilde{p}(s,t)\tilde{p}(s+x,t+y)dsdt \quad \textbf{(9)}$$

$$\begin{aligned} c_{\tilde{p}\tilde{p}}[m,n] = c_{\tilde{p}\tilde{p}}(mX,nY) &= \iint \tilde{p}(s,t)\tilde{p}(s+x,t+y)dsdt, \\ &= \frac{180^2}{(\pi D)^2}\iint \tilde{h}(s,t)\tilde{h}\left(s+\frac{180\,m}{\pi RD}, t+\frac{180\,n}{\pi RD}\right)dsdt, \\ &= \frac{180^2}{(\pi D)^2} c_{\tilde{h}\tilde{h}}\left(\frac{180m}{\pi RD}, \frac{180n}{\pi RD}\right) \end{aligned} \quad \textbf{(10)}$$

For the quick DBS method, each pixel must be toggled and swapped for the 10 states, and the updated error is then simplified, as shown in Equ. (11). If $\Delta\varepsilon < 0$ after toggling or swapping is carried out, this means that the cost function has decreased, indicating that the halftone image must be corrected, in which case toggling and swapping are defined as follows:

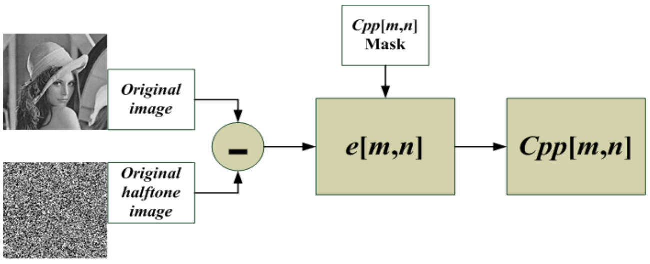
Toggle: $a_0 = \begin{cases} 1\ ,if & g[m_0,n_0]=0 \\ -1, if & g[m_0,n_0]=1 \end{cases};\ a_1 = 0$

Swap: $a_0 = \begin{cases} 1\ ,if & g[m_0,n_0]=0 \\ -1, if & g[m_0,n_0]=1 \end{cases};\ a_1 = -a_0$

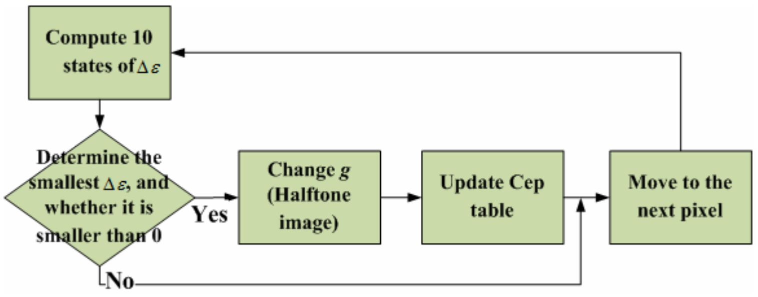
$$\Delta\varepsilon = (a_0{}^2 + a_1{}^2)c_{\tilde{p}\tilde{p}}[0,0] + 2a_0a_1c_{\tilde{p}\tilde{p}}[m_1-m_0,n_1-n_0] + 2a_0c_{\tilde{e}\tilde{p}}[m_0,n_0] + 2a_1c_{\tilde{e}\tilde{p}}[m_1,n_1] \quad \textbf{(11)}$$

$$c'_{\tilde{e}\tilde{p}}[m,n] = c_{\tilde{e}\tilde{p}}[m,n] + a_0c_{\tilde{p}\tilde{p}}[m-m_0,n-n_0] + a_1c_{\tilde{p}\tilde{p}}[m-m_1,n-n_1] \quad \textbf{(12)}$$

The 13×13 mask is defined as $c_{\tilde{p}\tilde{p}}$ [*m*,*n*]. For $c_{\tilde{e}\tilde{p}}$ [*m*,*n*], a table equivalent in size to the original image is created in practice. A table lookup-like method is used to compute $\Delta\varepsilon$ as the error *e*[*m*,*n*] will change when whenever the cost function decreases. Therefore, the $c_{\tilde{e}\tilde{p}}$ [*m*,*n*] table has to be constantly updated during the process, and the update method is shown in Equ. (12). The quick DBS process consists of two stages, as shown in Fig. 3.



(a) create a $c_{\tilde{e}\tilde{p}}$ [*m*,*n*] table


Compute 10 states of Δε
Determine the smallest Δε, and whether it is smaller than 0
Yes
No
Change g (Halftone image)
Update Cep table
Move to the next pixel

(b) iteration process 2 the process of depth intramode selection

Fig. 3. Quick DBS process

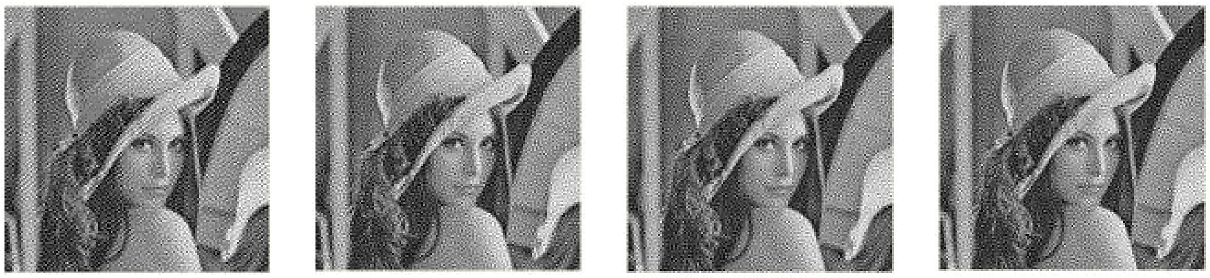
3 Proposed VLSI Architecture

3.1 Software Design

The implementation of the DBS method proposed in this work is carried out primarily through a hardware architecture design. The main function of the software algorithm is to verify whether the hardware results are consistent with the software results. Software implementation is carried out using Borland's C++ Builder (BCB) software, and the design is created primarily through the use of hardware computing methods. With respect to hardware design, floating-point numbers are dealt with by converting the HVS model's autocorrelation masks to integers. As halftone pixels only carry values between 0 and 255 (no intermediate values), approximation cannot be used to compare the hardware and software output values. Therefore, the same simplification process must be carried out for the software in order to verify whether the output data is consistent.

3.2 VLSI Architecture Design

Bit precision. The DBS method's computing process utilizes the HVS autocorrelation masks that are derived. The size of the masks is 13×13 [3], with the smallest value being 2.67471653922E-6 and the largest value being 0.042274635646005, and therefore, floating point numbers have to be computed. However, hardware computing can only handle 0 and 1 and not floating point numbers, and thus, this work proposes the level-shift method as a way to address this issue and to replace the hardware multiplier and thereby reduce hardware costs. Although there will be deviations in the results obtained using this approach when compared to the original results, these deviations may be ignored when there is a sufficient number of shifted bits. Furthermore, when too many bits are shifted, circuit area will increase alongside precision, hence data testing has to be carried out to determine the appropriate quantity of bits to shift and the optimal bit length [8]. After testing is carried out and hardware cost and precision issues are considered, it was decided that a 12 bit shift precision level will be implemented for the HVS autocorrelation masks examined in this study, as shown in Fig. 4. Additionally, 70 pixels will still carry a value of 0 after the autocorrelation mask values are shifted, thus computation loads can be reduced.



(a) 10 bits (b) 11 bits (c) 12 bits (d) 13 bits

Fig. 4. Hardware architecture for HVS mask bit shifting

Hardware architecture. The DBS method is implemented in this study with the aim of addressing problems with overly slow computing speeds, and thus, parallel computation is carried out within the same clock tick for all the points of the autocorrelation masks. Parallel computation can also be carried out for updated visual model error tables and halftone images. Fig. 5 shows the hardware architecture diagram for the DBS method.

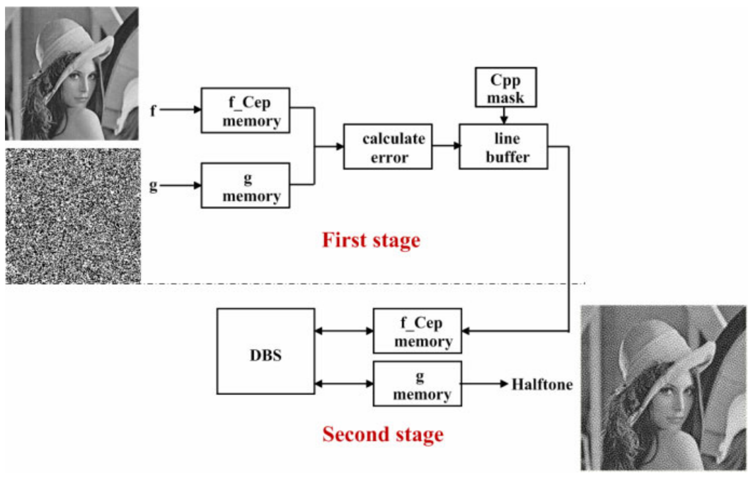


Fig. 5. Hardware architecture and flow process for the DBS method

In Fig. 5, memory *g* indicates the storage of the halftone image (one pixel is 1 bit), while the f_Cep memory indicates the memory used for the shared storage of the gray image (f) and the visual model error table (Cep). Since the gray image is used only once at the start of the computation carried out to determine the error value, the visual model error table will be stored in the f_Cep memory after the error value has been computed, so as to reduce memory usage. As the visual model error table is the result obtained after the visual model autocorrelation mask is computed, and the values of the autocorrelation mask are shifted left by 12 bits, each of the pixels in the f_Cep memory must be expressed using a bit length of at least 25 bits.

Output and input pin diagrams are presented in Fig. 6 and Table 1, in which the respective function of each pin is described. The data inputted for rb_Q is the result of the merging of the original halftone image and the gray image. The most significant bit is the value of the original halftone, while the remaining 8 bits are the values of the gray image. The merged bits diagram for rb_Q is shown in Fig. 7.

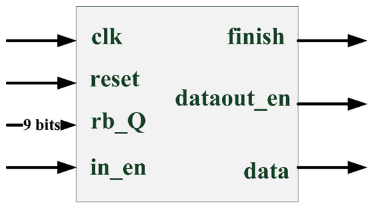


Fig. 6. Signal input and output pins for the DBS method

Table 1. Hardware pin functions for the DBS method

Signal Name	I/O	Width	Description
clk	input	1 bit	posedge clk
reset	input	1 bit	negedge reset
rb_Q	input	9 bits	data(g and f) in
in_en	input	1 bit	input enable
finish	output	1 bit	DBS finish
dataout_en	output	1 bit	data output enable
data	output	1 bit	halftone pixel value

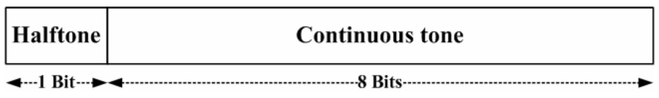


Fig. 7. Merged bits diagram

Use of a line buffer to create a visual model error table. Before the DBS method is carried out, the visual model error table is first created. This table is created by obtaining the convolution integrals of the

HVS autocorrelation mask (Cpp) and the error between the continuous color image and the original halftone image. After the convolution integral computations are completed for the autocorrelation mask and error value, the table can then be directly stored in f_Cep memory. The line buffer approach can be used to save on memory space, and each pixel of the storage error's line buffer will have a bit depth of 12 bits. For the line buffer design, the size of the mask (the mask in this study is 13×13 [3]) has to match the image width (256) of the mask for which convolution integral computations are carried out. Therefore, a 12×256 line buffer and a register measuring 12 bits in length can be designed, as shown in Fig. 8. As the range required for the mask is 13×13, and the side length of the line buffer architecture is 13 bits, the wide section is sufficiently long and only the bottommost register is 12 bits in length. This is because the read-in values can be directly computed when the line buffer is fully filled with data, which reduces the space needed by one register.

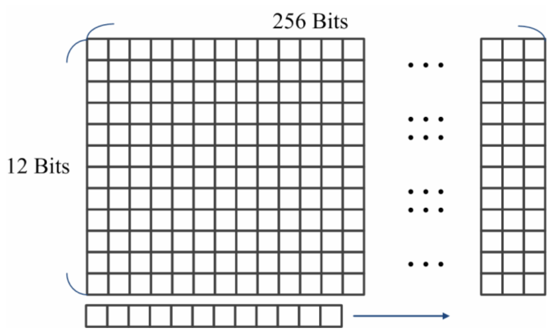


Fig. 8. Line buffer architecture

After every mask multiplication, the mask has to be shifted right in order for the next round of computation to occur. At this point, the value in the upper left corner of the previous masking range will no longer be used. Therefore, the values in the leftmost column are moved up by one cell after mask multiplication is completed. Following this, the bottommost register is shifted to the left by one cell, thus allowing new incoming values to be stored together. By doing so, the mask is continuously being computed in a rightward direction, and once mask multiplication has been completed for the rightmost side, the values in the entire mask range can be simultaneously moved upward by one cell, bringing them back to the leftmost side of the line buffer. Computations are repeated in this manner until the visual model error table is completed. With this architecture and time sequence design, masking can be carried out smoothly, and the architecture does not have to store the original image that indicates the error between the continuous color and original halftone images, which a reduction of 62452×12 bits in register usage.

Core computation for DBS method. After the visual model error table is completed, the core computation for the DBS method can then be carried out. This process involves corrections being constantly made to the halftone image to make it more closely resemble the continuous gray image. After the halftone image is corrected, the 13x13 values surrounding the positions in the visual model error table that correspond to the shifted pixels in the halftone image must also be updated. The core architecture of the DBS method can be divided into five processes, as shown in Fig. 9. The original image has 256×256 pixels, implying that the core architecture of the DBS method must be executed 65,536 times. Usually, iteration has to be carried out a dozen times or more for this method, making it impossible for execution to occur in real time. In the hardware architecture designed for this study, the updating of the visual model error table and halftone image occur simultaneously, the comparison results for the 10 states of the smallest determined dE are also computed simultaneously. Compared to the software approach, in which determination is carried out unit by unit and processing is executed line by line, the hardware approach utilizes a parallel architecture design that increases processing speeds. When the cost function stops decreasing, the iteration process is complete and the halftone image is outputted.

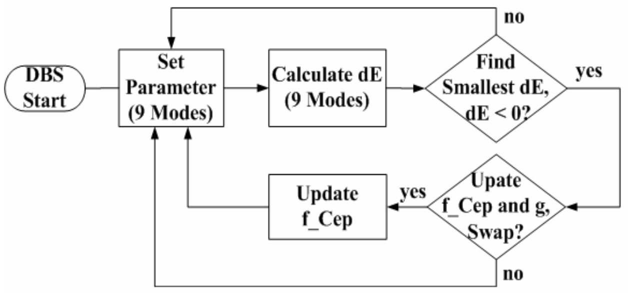
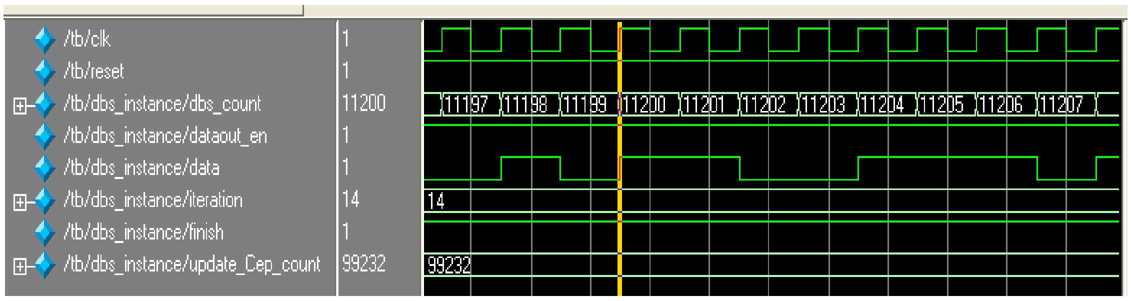


Fig. 9. Core architecture of DBS method

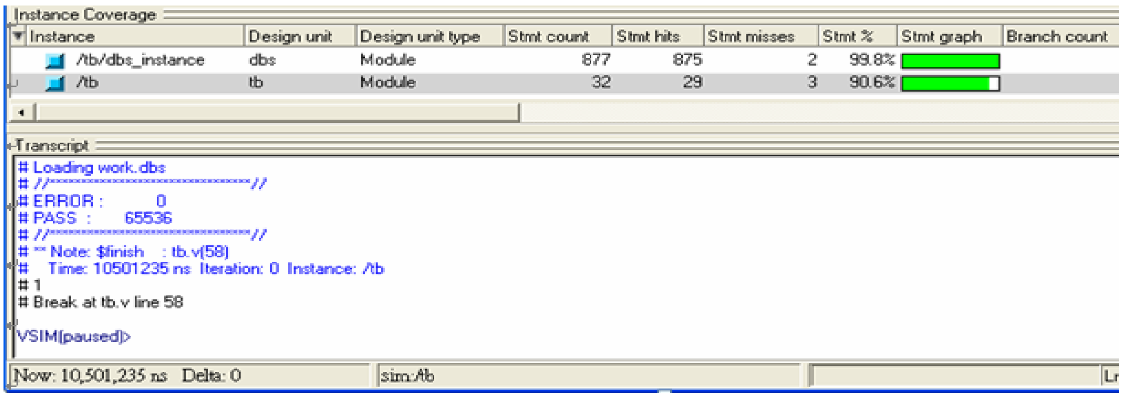
4 Experimental Analysis and Comparison

In this work, a 256×256 original gray image was simulated using Borland's C++ Builder (BCB) software, and the results indicate that the DBS method achieved an execution time of 125ms, required 14 rounds of iteration, and updated the Cep table 99,232 times.

With respect to the hardware, the Verilog Hardware Description Language (HDL) was used to design the architecture and the ModelSim software was used to simulate the verification process. As shown in Fig. 10, the a 256×256 gray image was inputted and the results of the waveform simulation was outputted the hardware. In Fig. 10(a), the DBS process is completed when "finish" is indicated as "high"; "dataout_en" indicates whether the output value is effective; "data" indicates the halftone pixel values being outputted; "dbs_count" marks the pixels up to the 11,200th outputted pixel. In Fig. 10, the number of iterations is indicated as 14 and the updated visual model error table (Cep) was updated 99,232 times.


```
/tb/clk                              1
/tb/reset                            1
/tb/dbs_instance/dbs_count           11200   11197 11198 11199 11200 11201 11202 11203 11204 11205 11206 11207
/tb/dbs_instance/dataout_en          1
/tb/dbs_instance/data                1
/tb/dbs_instance/iteration           14      14
/tb/dbs_instance/finish              1
/tb/dbs_instance/update_Cep_count    99232   99232
```

(a) Hardware results


```
Instance Coverage
Instance            Design unit  Design unit type  Stmt count  Stmt hits  Stmt misses  Stmt %  Stmt graph  Branch count
  /tb/dbs_instance  dbs          Module                   877        875            2   99.8%
  /tb              tb           Module                    32         29            3   90.6%

Transcript
# Loading work.dbs
# //******************************//
# ERROR :        0
# PASS  :     65536
# //******************************//
# ** Note: $finish    : tb.v(58)
#    Time: 10501235 ns  Iteration: 0  Instance: /tb
# 1
# Break at tb.v line 58

VSIM(paused)>

Now: 10,501,235 ns   Delta: 0        sim:/tb
```

(b) Software results

Fig. 10. ModelSim simulated output waveform

5 Conclusion

This work analyzed a new VLSI architecture design for the algorithms of the DBS halftone imaging method. The proposed hardware architecture addresses internal memory issues through the utilization of a line buffer with a memory usage of 62452×12 bits and the implementation of a parallel architecture design that enables the visual model error table and halftone image to be updated simultaneously. With respect to the smallest determined error (dE), the comparison results for the 10 states (nine determination formulas) are also computed simultaneously as the 65,536 (256×256) pixels of the original image must be determined and iterated 14 times. Therefore, the execution time for this approach is shorter by 65536×14×8 clock ticks compared to most software algorithms and the processing speed for halftone images is also increased. Moreover, this hardware architecture can also effectively address some of the issues associated with halftone imaging technology, such as the number of required iterations and the updating of the visual model error table.

Acknowledgements

The authors would like to thank the anonymous reviewers of their paper for the many helpful suggestions. This work was supported by the Ministry of Science and Technology of Taiwan, R.O.C. under grant number MOST 106-2221-E-034-019-. Then, this work was supported by the Education Scientific Research Project of Young Teachers of Fujian Province under grant number JAT170577, the climbing project of Longyan University under grant number LQ2015031, Fujian university's key lab. of bigdata mining and application and Collaborative innovation project of Longyan University(L. Zhang). Finally, this work was supported by the Longyan Municipal Bureau of Finance under grant number 2017LY75.

References

[1] M. Analoui, J.P. Allebach, Model based half-toning using direct binary search, Proceeding of SPIE 1666(1992) 96-108.

[2] J.-M. Guo, C.-Y. Lin, Parallel and element-reduced error-diffused block truncation coding, IEEE Transactions on Communications 58(5)(2010) 1667-1673.

[3] S.H. Kim, J.P. Allebach, Impact of HVS models on model-based halftoning, IEEE Transactions on Image Processing 11(3)(2002) 258-269.

[4] G.J.P. Barten, Contrast Sensitivity of the Human Eye and Its Effects on Image Quality, HV Press, Knegsel, Netherlands, 1999.

[5] D.J. Lieberman, J.P. Allebach, Efficient model based halftoning using direct binary search, IEEE International Conference on Image Processing 1(1997) 775-778.

[6] D.J. Lieberman, J.P. Allebach, A dual interpretation for direct binary search and its implications for tone reproduction and texture quality, IEEE Transactions on Image Processing 9(11)(2000) 1950-1963.

[7] J.-M. Guo, C.-H. Hsia, C.-H. Chang, JPEG false contour reduction using error diffusion, Information Processing Letters 15(3)(2015) 403-407.

[8] C.-H. Hsia, J.-S. Chiang, J.-M. Guo, Memory-efficient hardware architecture of 2-D dual-mode lifting-based discrete wavelet transform, IEEE Transactions on Circuits and Systems for Video Technology 23(4)(2013) 671-683.