

PARALLEL INVERSE HALFTONING BY LOOK-UP TABLE (LUT) PARTITIONING

Umair F. Siddiqi* and Sadiq M. Sait**

Department of Computer Engineering
King Fahd University of Petroleum & Minerals
Saudi Arabia

الخلاصة:

تحتاج طريقة تجزئة جدول البحث للون النصفى العكسي إلى خطوات حساب أقل وتعطي نتائج جيدة . وتستخدم الطريقة جدول بحث واحداً يُخزَّن في ذاكرة للقراءة فقط ، ويحتوى الجدول على قيم لتخفيف اللون سابقة الحساب (المستوى الرمادي) لعمليات اللون النصفى العكسي . وهذا البحث يقترح طريقة حساب متوازية لإنجاز اللون النصفى العكسي ، وذلك بتجزئة جدول البحث المنفرد إلى عدد من جداول البحث المصغرة ، وحتى ك (حيث $K > N$) نقطة ، يمكن استحضارها من الصورة ذات اللون النصفى ، وكذلك يمكن جلب قيمها الخاصة بتخفيف اللون أنبأ من جداول بحث مصغرة .

و عملية التوازي تزيد سرعة اللون النصفى العكسي ك من المرات ، بينما يظل مجموع المدخلات بجميع جداول البحث المصغرة مساوياً لمدخلات جدول البحث المنفرد المستخدم بالطريقة التسلسلية لجدول البحث . وهناك احتمال لانخفاض جودة الصورة بسبب فقد النقاط في أثناء الاستحضار المتوازي ، ويُعزى ذلك إلى عدم التمكن من جلب قيم تخفيف اللون في الدورة نفسها بسبب استحضار قيم أخرى لتخفيف اللون من جداول البحث المصغرة .

ويحتاج التطبيق الكامل لطريقة الحساب إلى جهازى برمجة منطقية مركبة للجزء الخاص بالحساب ، ووحدة ذاكرة خارجية ذات عناوين ، ووحدات ذاكرة وصول عشوائي ساكنة لتخزين جداول البحث المصغرة .

* Address for correspondence:

E-mail: umaith@nhupm.edu.sa

KFUPM Box: 673

Department of Computer Engineering

King Fahd University of Petroleum & Minerals

Dhahran-31261, Saudi Arabia

Telephone: +966-3-860 1099

Fax: +966-3-860 3955

** E-mail: sadiq@kfupm.edu.sa

Paper Received 12 December 2006; Revised 23 January 2008; Accepted 4 June 2008

ABSTRACT

The Look-Up Table (LUT) method for inverse halftoning is not only computation-less and fast but yields good results. The method employs a single LUT that is stored in a ROM and contains pre-computed contone (gray level) values for inverse halftone operation. This paper proposes an algorithm that can perform parallel inverse halftone operations by partitioning the single LUT into N smaller Look-Up Tables (s-LUTs). Therefore, up to k ($k \leq N$) pixels can be concurrently fetched from the halftone image and their contone values fetched concurrently from separate s-LUT. Obviously, this parallelization increases the speed of inverse halftoning by up to k times. In this proposed method, the total entries in all s-LUTs remain equal to the entries in the single LUT of the serial LUT method. Some degradation in image quality is possible due to pixel loss during parallel fetching. This is because some contone values cannot be fetched in the same cycle because some other contone value is being fetched from that s-LUT. The complete implementation of the algorithm requires two CPLDs (Complex Programmable Logic Devices) for the computational portion, external content addressable memories (CAM) and static RAMs to store s-LUTs.

Key words: inverse halftoning, hardware implementation, look-up table inverse halftoning, complex programmable logic devices (cpld), image processing, parallelizing

PARALLEL INVERSE HALFTONING BY LOOK-UP TABLE (LUT) PARTITIONING

1. INTRODUCTION

The process of rendition of continuous tone pictures on media on which only two levels can be displayed is defined as halftoning [1]. The problem has gained importance since the time of the printing press when attempts were made to print images on paper by adjusting the size of dots according to the local print intensity. This process is termed as analog halftoning. With the availability and adoption of bi-level devices such as fax machines and plasma displays, digital halftoning has become important [2]. The input to a digital halftoning system is a gray level image in which pixels have more than two levels (*e.g.*, 256 levels), and the result of the halftoning process is an image that has only two levels *i.e.*, 1 or 0. Inverse halftoning on the other hand is the reconstruction of gray level images from halftone images. Inverse halftone operation finds application in areas where processing is required on printed images. The images are first scanned, inverse halftoned, and then operations like zooming, rotation, and transformation are applied. Standard compression techniques cannot process halftones directly and therefore inverse halftoning is required before compression of printed images can be performed [1].

Look-Up Table (LUT) inverse halftoning is a fast and low computation method [3]. LUT inverse halftoning was first introduced by Netravali and Bowen [4], but requires some information to be known that is not always available for halftone images. Subsequently Ting and Riskin [5] proposed another LUT method but did not target image quality. In the recent past, a computation free LUT method was proposed by Mese and Vaidyanathan [1, 3]. This method provides fast inverse halftoning with good image quality, and can be applied on several different halftones. Two more methods for LUT inverse halftoning [6, 7] were suggested by Kuo-Liang Chung *et al.* and P. C. Chang *et al.* which give better image quality but are not completely computation free. In addition to Look-Up Table (LUT) access, add operations need to be performed.

In the Mese *et al.* method, one template that consists of the pixel to be inverse halftoned, and pixels in its neighborhood, are fetched from the halftone image in a p -bits ($p=17, 21, 22$) vector and used to form the address for the LUT. Its pre-computed contone value is fetched from this address of the LUT. However, this method is serial and is able to inverse halftone only one template at a time. In this paper, we present an algorithm that can perform parallel inverse halftone operations by partitioning the single LUT of the Mese *et al.* method into N smaller Look-Up Tables (s-LUTs). The N s-LUTs contain total entries equal to the entries in the single LUT of the serial LUT method. In this way, the proposed algorithm can provide significant advantages in speed of inverse halftone operation and at the same time provide a saving in memory requirements. In the proposed algorithm, ' k ' templates are concurrently fetched from the halftone image and their contone values are obtained through s-LUTs.

This paper is organized as follows. First the serial LUT method is described. Then the parallelization of LUT method for inverse halftoning is discussed in detail, which basically employs partitioning the LUT based on some criteria. This is followed by the simulation of the proposed algorithm and discussion about its performance. In the last section, implementation details of the proposed algorithm using CPLDs are discussed.

2. LOOK-UP TABLE (LUT) METHOD FOR INVERSE HALFTONING

In the LUT method for inverse halftoning a template represented by ' t ' is a group of pixels that consists of pixel to be inverse halftoned and the pixels in its neighborhood. The LUT method uses three types of templates namely: *16pels*, *19pels* and *Rect*. The *16pels* template consists of 17-pixels, *19pels* consists of 20-pixels and *Rect* consists of 22 pixels. The templates are fetched from the halftone image in a raster-scan style, *i.e.*, from left to right, and from top to bottom. One pixel with surrounding ones (so called a template (t)) is fetched and inverse halftoned before the next template is fetched. The Look-Up Table (LUT) stores pre-computed contone values of a large number of templates. The templates for storage in the LUT are obtained from a training set of images that comprise halftone images and corresponding continuous tone images. The templates are fetched from the halftone images and their contone values are fetched from corresponding continuous tone images. When a template occurs more than once, then its contone value is the mean of all contone values that correspond to that template in the training set. The inverse halftone operation is performed in such a way that a template (t) is fetched from the halftone image and is sent to the Look-Up Table (LUT). If the LUT has the stored contone value for the template (t), it returns it, otherwise the template (t) goes through any one of these methods: (a) Low Pass Filtering; or (b) Best Linear Estimator [1]. When the same halftone algorithm is used in training set images and the images going through inverse halftone operation, then all templates always find their corresponding contone value in the LUT, and consequently, this method becomes completely computation free. The LUT method for inverse halftoning can also be applied to color halftones, where a separate LUT exists for color planes R , G , and B .

3. PARALLEL LOOK-UP TABLE (LUT) INVERSE HALFTONING

In order to perform parallel LUT inverse halftoning, two or more templates should be fetched from the halftone image and LUT (Look-Up Table) inverse halftone operation is applied to them at the same time. The main problems in parallelizing LUT method for inverse halftoning are the following:

- The Look-Up Table (LUT) is composed of a single memory block that does not allow simultaneous access to more than one location. Therefore, parallel templates cannot fetch their contone values at the same time.
- If the LUT method for inverse halftoning is parallelized as it is then the memory requirements grow very large because one needs to store one template (t) for each template that is fetched in parallel.

In the subsequent section we present an algorithm to parallelize the LUT method for inverse halftoning while solving the above problems.

4. ALGORITHM TO PERFORM PARALLEL LUT INVERSE HALFTONING

This section shows the algorithms that can perform parallel inverse halftone operation by enhancing the serial LUT method of Mese and Vaidyanathan [3]. In the proposed algorithm, N smaller Look-Up Tables (s-LUTs) are used in place of the single LUT of the serial LUT method. The proposed algorithm also introduces a circuitry that can distinguish k templates that are concurrently fetched from the halftone image through unique numbers. As a result of these two modifications, k templates can be fetched concurrently and go through parallel inverse halftone operation using N s-LUTs and therefore their contone values can be obtained simultaneously. The proposed parallel inverse halftoning using s-LUTs consists of two steps: (1) an algorithm to generate ' N ' smaller Look-Up Tables (s-LUTs); and (2) an algorithm to send ' k ' concurrently fetched templates to distinct s-LUTs. In the rest of this section the algorithms are described in detail.

4.1. Idea Behind the Proposed Algorithm

The algorithm proposed to perform parallel inverse halftone operation is based on the idea of partitioning the single LUT into N smaller Look-Up Tables (s-LUTs). The partitioning can be done linearly or can use any sophisticated technique. In linear partitioning the contents from the training set are assigned to s-LUTs based on some fixed criteria like equal number of contents in all N s-LUTs. This approach has the problem that during inverse halftone operation it becomes difficult to estimate which template value exists in which s-LUT. The algorithm proposed in this paper instead partitions the LUT into N s-LUTs by using a new approach. The new approach is based on the observation that in some halftone images, adjacent, *i.e.*, either top-bottom, or left-right template, values differ from each other in the number of ones present in them. In this paper we define a function that takes XOR between the fetched template and m , where m is the mean of all template values present in the training set. Then the bits in the XOR result are added to calculate the number of ones. At this point a unique result is obtained for each concurrently fetched template, *i.e.*, k unique values are obtained. However, their values vary from 0 to $2^p - 1$, whereas number of s-LUTs is N . In the next step, mod N operation is applied and numbers in range from 0 to $N-1$ are obtained for all concurrently fetched templates. The graph in Figure 1 shows the percentage of times this approach is successful to distinguish concurrently fetched templates. The mod N operation is computation free when N is an exponent of 2, *i.e.*, 2, 4, 8, 16, 32, or etc., and the result is the $\log_2 N$ least significant bits.

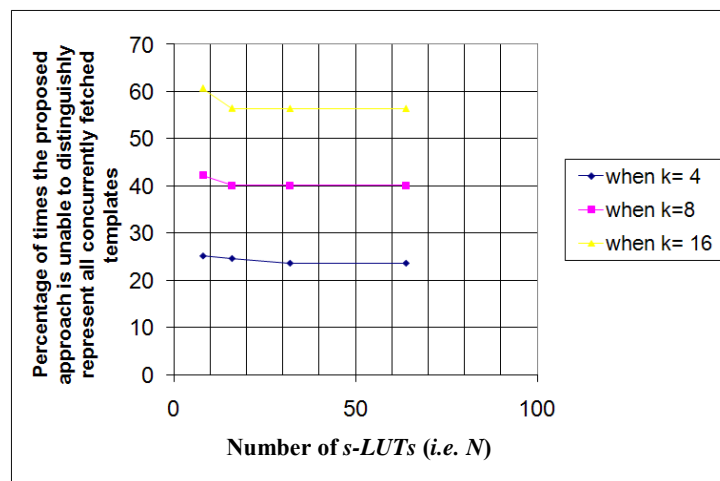


Figure 1. Graph showing performance of proposed approach to distinguish concurrently fetched templates

The N s-LUTs will be stored in N external memories, and templates fetched from the halftone image act as input addresses to memories. Distribution of templates among N s-LUTs should be uniform so that memories of equal size can be utilized, however, when s-LUTs do not have equal size than large s-LUTs can be stored in more than one independent memory block and in that case (*number of memories*) $> N$. Two other approaches that can be also be used are: (1) to add the bits in the templates and then take mod N ; and (2) directly take mod N of the fetched template. However, first taking XOR with m yields the best image quality among them, therefore the algorithm proposed in this paper uses only this approach.

4.2. Algorithm to Generate N Smaller Look-Up Tables (s-LUTs)

N smaller Look-Up Tables (s-LUTs) must be generated before inverse halftone operation is performed, similar to the procedure for the serial LUT method. The s-LUTs are numbered from 0 to $N-1$, where N must be an exponent of 2, *i.e.*, 2, 4, 8, 16, *etc.* The algorithm is shown in Figure 2. It starts by building a 'Training_set', which consists of continuous tone images and their halftone versions. In Step 2, a template represented by 't' is fetched from the halftone image. In Step 3, first the fetched template t is XORed with m , where m is the mean of all template values present in the training set. Then bits in the obtained result are added and finally its mod N value is obtained from the least significant $\log_2 N$ bits. The result now obtained is the result of Step 3. In the next step, template 't' is sent to s-LUT that has same number as the result returned in Step 3. Now the above procedure from Step 2 to Step 4 is repeated by fetching another template from the training set and this continues until all templates in the training set are fetched and stored in s-LUTs.

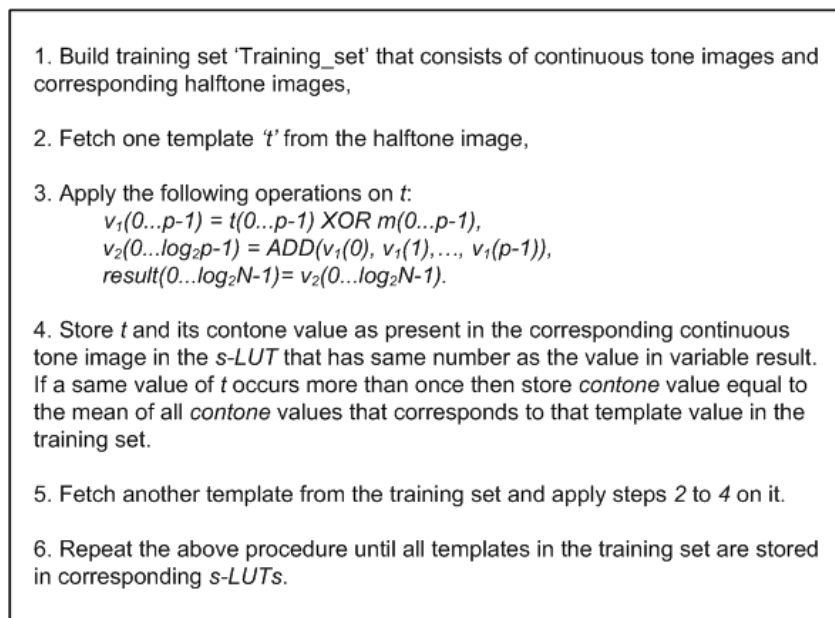


Figure 2. Algorithm to generate smaller Look-Up Tables (s-LUTs)

4.3. Algorithm to Send 'k' Concurrently Fetched Templates to Distinct s-LUTs

This algorithm performs the task of assigning unique numbers in range from 0 to $N-1$ to k concurrently fetched templates and then sending them to distinct s-LUTs using their unique numbers. In this way it can perform parallel inverse halftone operation using s-LUTs. The algorithm is shown in Figure 3. It starts by fetching k templates from the halftone image in which the templates are numbered from 1 to k . Then in Step 3, mod- N operation is performed on template. In Step 4, if two or more templates have same value returned from Step 3 then among them only the template that has the highest number assigned to it in Step 2 is kept and others templates that returned the same result in the Step 3's are discarded. The templates that are not discarded are now sent to s-LUTs that have same numbers as their values returned in Step 3. In Step 6, *contone* values to templates that were discarded in Step 4 or the templates that do not find their *contone* values in their s-LUTs are assigned by copying *contone* values from their neighbors. Finally *contone* values of all k fetched templates are delivered to the output. This process repeats until all templates present in the halftone image are inverse halftoned. This algorithm is pipelined, therefore, each step can be performed in parallel on different data inputs and new k templates can be fetched in every clock cycle from the halftone image.

1. Concurrently fetch k templates represented by t_0, t_1, \dots, t_{k-1} from the halftone image,
2. Assign numbers to the fetched templates from 1 to k ,
3. Apply the following operations to each template simultaneously:

$$u_i(0..p-1) = t_i(0..p-1) \text{ XOR } m(0..p-1),$$

$$w_i(0..\log_2 p-1) = \text{ADD}(u_i(0), u_i(1), \dots, u_i(p-1)),$$

$$v_i(0..\log_2 N-1) = w_i(0..\log_2 N-1),$$
 where p is the number of bits in the template, and $i= 0$ to $k-1$.
4. if $v_i=v_j$ where $i \neq j$ and $i > j$, then discard if v_i , both i and j varies from 0 to $k-1$.
5. Send templates whose results from step 3 are not discarded to s -LUTs that has same number as the values returned in step 3. In this step templates obtained their contone values from s -LUTs.
6. Assign *contone* values to templates that were not send to s -LUTs in step 5 or to templates whose *contone* value are not found in the corresponding s -LUT by copying *contone* values of template that has the nearest higher number assigned to it in step 2 and whose step 3's result was not discarded in step 4.
7. Yield *contone* values of input templates.

Figure 3. Algorithm to perform parallel inverse halftoning using s -LUTs

5. SIMULATION

This section shows some simulation results of the proposed algorithm implemented in Java programming language. The simulation process starts by building a training set of 17 gray level and their corresponding halftone images. Then the value of N is chosen and s -LUTs are generated. The parallel inverse halftone operation is performed by setting different values of k . In this section, first the results of generating s -LUTs are shown and then some halftone images that are not present in the training set are inverse halftoned and are shown with their image quality in terms of Peak Signal to Noise Ratio (PSNR).

5.1. Generation of s -LUTs

The s -LUTs are generated using the training set and partitioning of templates among N s -LUTs is shown in Figures 4 and 5. Figure 4 shows the partitioning when $N= 8$ and Figure 5 shows the partitioning when $N=16$. It is shown that each s -LUT stores almost equal number of contents when $N= 8$, and when $N= 16$ large deviation in required s -LUT sizes is observed.

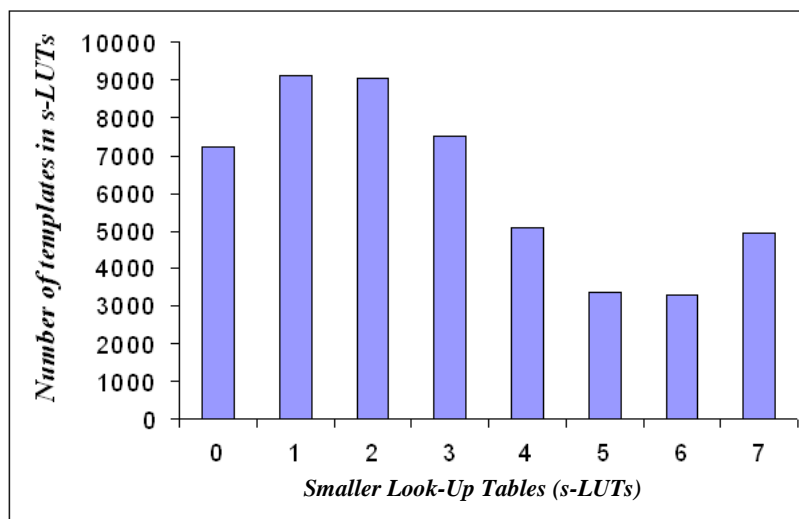


Figure 4. Distribution of templates to s -LUTs (when $N=8$)

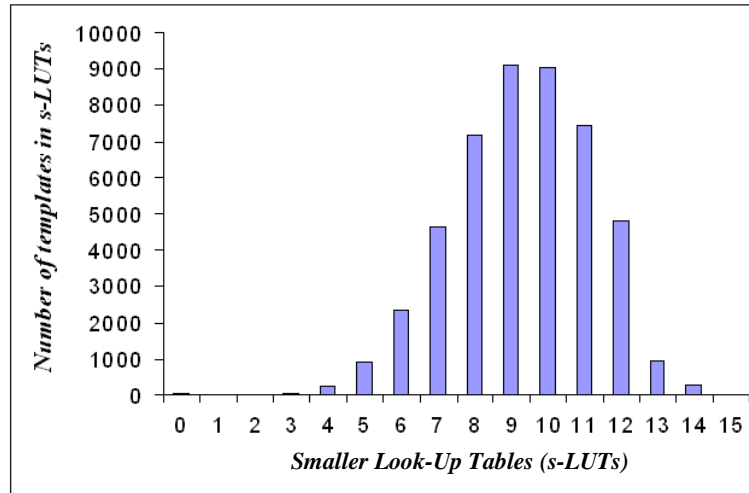


Figure 5. Distribution of templates to s-LUTs (when N=16)

5.2. Parallel Inverse Halftoning

This section shows the simulation of the proposed algorithm to perform parallel inverse halftone operations using s-LUTs. The simulation shows inverse halftone operation for different values of k and N . The graph in Figure 6 shows average image quality when compared to quality of images obtained from the proposed algorithm in terms of PSNR. The same graph also shows curves drawn for different values of k and their y-axis values *i.e.*, image quality varies with increase in the value of N . However, N should be an exponent of 2 *i.e.*, 2, 4, 8, 16, *etc.* The results show an average of results obtained from images: Boat, Peppers, and Clock. Some sample images obtained from the serial LUT method and from proposed algorithm are shown in Figures 7 to 15, along with their original continuous tone versions.

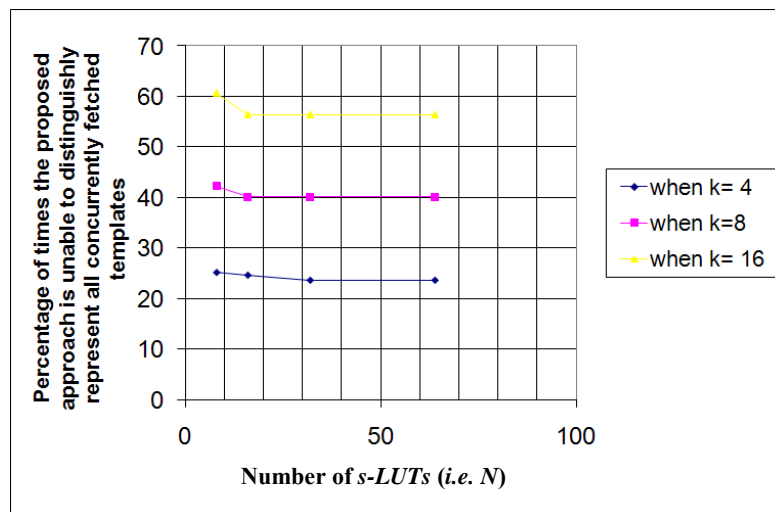


Figure 6. Performance of the proposed algorithm in terms of image quality for different values of k and N



Figure 7. Original continuous tone image named 'Peppers'



Figure 8. Peppers obtained from serial LUT method, PSNR = 29.4154 dB



Figure 9. Peppers obtained from inverse halftone operation using proposed algorithm with $k = 4$ and $N = 8$, PSNR = 29.2605 dB



Figure 10. Original continuous tone image named 'Clock'

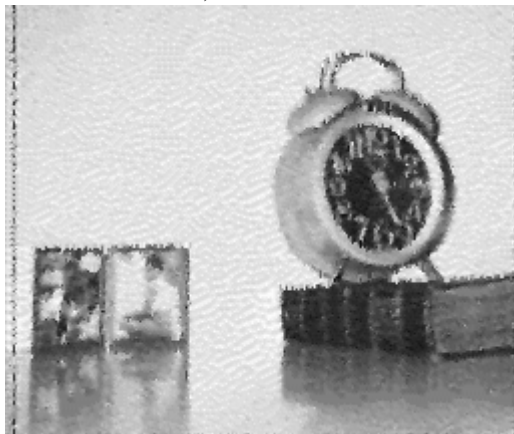


Figure 11. Clock obtained from serial LUT method, PSNR = 30.1681 dB

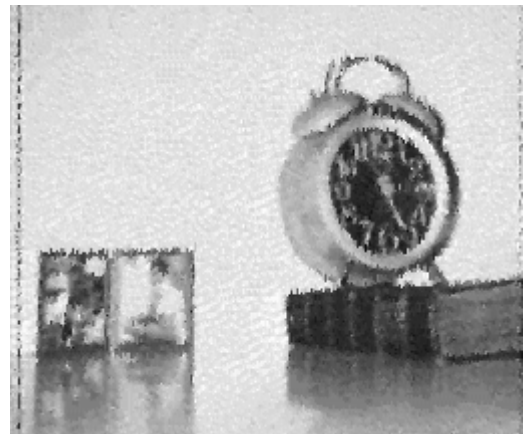


Figure 12. Clock obtained from inverse halftone operation using proposed algorithm with $k = 4$ and $N = 8$, PSNR = 30.0846 dB



Figure 13. Original continuous tone image named 'Boat'



Figure 14. Boat obtained from serial LUT method, PSNR = 28.7071 dB



Figure 15. Boat obtained from inverse halftone operation using proposed algorithm with $k = 4$ and $N = 8$, PSNR = 28.5449 dB

6. LOGIC CIRCUIT DESIGN

A logic circuit that can implement the proposed algorithm with parameters $k=4$ and $N=8$ has been designed. The target platform is Field Programmable Gate Array (FPGA) devices. The circuit is divided into blocks and each block can work independently on different inputs. The connections among different blocks are shown in Figure 16. In the following paragraphs we discuss the details of the various blocks.

Block 1: In this stage, four templates $I_0, I_1, I_2,$ and I_3 are fetched from the halftone image and stored in registers $t_0, t_1, t_2,$ and t_3 respectively. The Boolean equations representing the logic in this block are shown below:

$$t_0(0...p-1) = I_0(0...p-1), t_1(0...p-1) = I_1(0...p-1), t_2(0...p-1) = I_2(0...p-1), t_3(0...p-1) = I_3(0...p-1) \quad (6.1)$$

Block 2: In this block bits in each template are added using Carry Save Adder (CSA) Tree. The Boolean equations representing operations in this block are shown below and the CSA tree for $N=8$ and $p=20$ is shown in Figure 17:

$$S_i = \text{CSA_TREE}(t_i(0...p-1)), \text{ where } i=0, 1, 2, \& 3. \quad (6.2)$$

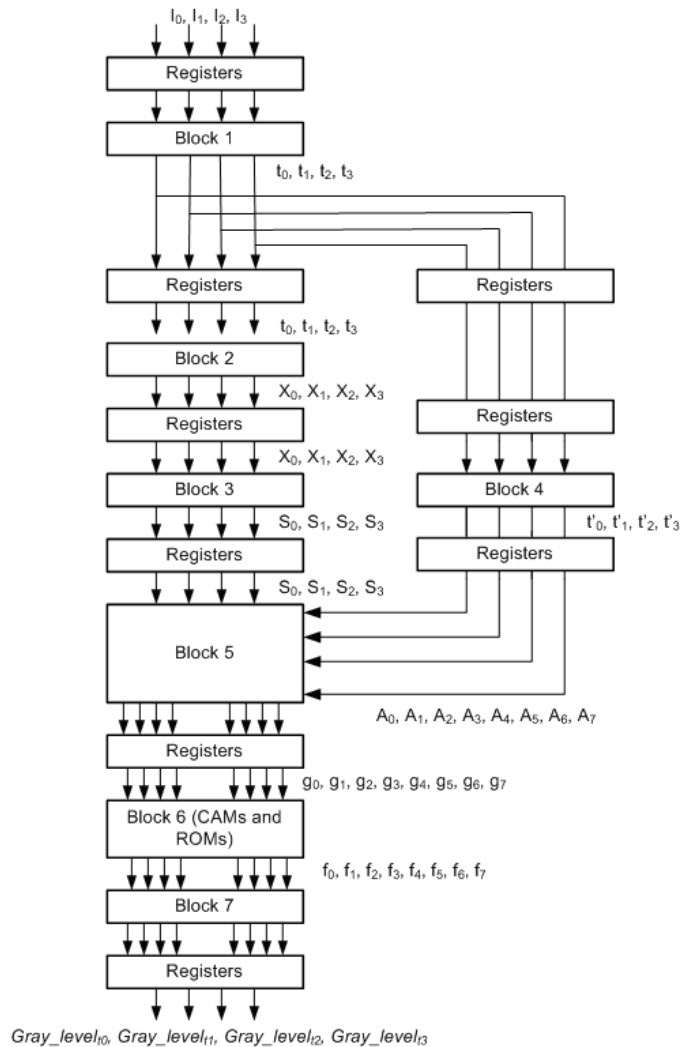


Figure 16. Illustration of connections among blocks that comprise the IC to implement the proposed algorithm

Block 3: In this block, templates t_0 , t_1 , t_2 , and t_3 are appended with sequence numbers 001, 010, 011, and 100 respectively. The Boolean expressions representing operations in this block are:

$$\begin{aligned}
 t_0'(0...p+2) &= t_0(0...p-1) \& 001, & t_1'(0...p+2) &= t_1(0...p-1) \& 010, \\
 t_2'(0...p+2) &= t_2(0...p-1) \& 011, & t_3'(0...p+2) &= t_3(0...p-1) \& 100.
 \end{aligned}
 \tag{6.3}$$

Block 4: This block consists of four 1×8 multiplexers. The Boolean expressions that represent the logic in this block are:

$$\begin{aligned}
 A_i[0](0...p+2) &\leftarrow \overline{slut_i(2)} \cdot \overline{slut_i(1)} \cdot \overline{slut_i(0)} \cdot (t_i'(0...p+2)), \\
 A_i[1](0...p+2) &\leftarrow \overline{slut_i(2)} \cdot \overline{slut_i(1)} \cdot slut_i(0) \cdot (t_i'(0...p+2)), \\
 A_i[2](0...p+2) &\leftarrow \overline{slut_i(2)} \cdot slut_i(1) \cdot \overline{slut_i(0)} \cdot (t_i'(0...p+2)), \\
 A_i[3](0...p+2) &\leftarrow \overline{slut_i(2)} \cdot slut_i(1) \cdot slut_i(0) \cdot (t_i'(0...p+2)), \\
 A_i[4](0...p+2) &\leftarrow slut_i(2) \cdot \overline{slut_i(1)} \cdot \overline{slut_i(0)} \cdot (t_i'(0...p+2)), \\
 A_i[5](0...p+2) &\leftarrow slut_i(2) \cdot \overline{slut_i(1)} \cdot slut_i(0) \cdot (t_i'(0...p+2)), \\
 A_i[6](0...p+2) &\leftarrow slut_i(2) \cdot slut_i(1) \cdot \overline{slut_i(0)} \cdot (t_i'(0...p+2)), \\
 A_i[7](0...p+2) &\leftarrow slut_i(2) \cdot slut_i(1) \cdot slut_i(0) \cdot (t_i'(0...p+2)),
 \end{aligned}
 \tag{6.4}$$

Block 5: This block consists of eight 4×1 multiplexers that are connected to s-LUTs. The Boolean expressions that represent logic operations in this block are shown below:

$$\begin{aligned}
 g_i(0..p+2) &\leftarrow (A_i[i](p) + A_i[i](p+1) + A_i[i](p+2)) \cdot A_i[i](0..p+2) + \overline{(A_i[i](p) + A_i[i](p+1) + A_i[i](p+2))} \cdot \\
 &(A_3[i](p) + A_3[i](p+1) + A_3[i](p+2)) \cdot A_3[i](0..p+2) + \overline{(A_i[i](p) + A_i[i](p+1) + A_i[i](p+2))} \cdot \\
 &\overline{(A_3[i](p) + A_3[i](p+1) + A_3[i](p+2))} \cdot (A_2[i](p) + A_2[i](p+1) + A_2[i](p+2)) \cdot A_2[i](0..p+2) + \\
 &\overline{(A_i[i](p) + A_i[i](p+1) + A_i[i](p+2))} \cdot \overline{(A_3[i](p) + A_3[i](p+1) + A_3[i](p+2))} \cdot \overline{(A_2[i](p) + A_2[i](p+1) + A_2[i](p+2))} \cdot \\
 &(A_1[i](p) + A_1[i](p-1) + A_1[i](p+2)) \cdot A_1[i](0..p+2)
 \end{aligned}
 \tag{6.5}$$

In the above equation, $i = 0$ to 7 and $i = 0$ refers to first multiplexer, $i=1$ refers to second multiplexer and so on. The output g_0 is from first multiplexer, g_1 is from second multiplexer and so on.

Block 6: In this block, templates fetch their gray level values from s-LUTs. In terms of hardware, it contains the implementation of eight smaller Look-Up Tables (s-LUTs) using Content Addressable Memory (CAM) and Read Only Memory (ROM) pairs. A combination of CAM–ROM is used because each s-LUT stores a very small fraction of values out of 2^p possible values, when templates are p -bits wide. The block diagram in Figure 18 shows the implementation of one s-LUT. The CAM stores the templates that are assigned to the s-LUT and ROM stores the gray level values. The Boolean equations illustrating the operations in this block are shown below:

$$\begin{aligned}
 \text{number of entries in a smaller Look - Up Table (sLUT)} &= 2^d - 1 \\
 \text{number of grey - levels} &= 256 \text{ i.e. } 8 - \text{bits} \\
 x_i(0..d-1) &\leftarrow \text{CAM}_i(g_i(0..p-1)), \\
 c_i(0..7) &\leftarrow \text{ROM}_i(x_i(0..d-1)) \\
 f_i(0..p+2) &\leftarrow g_i(0..p+2)
 \end{aligned}
 \tag{6.6}$$

In the above expressions, $i = 0$ for s-LUT number 0, $i=1$ for s-LUT number 1 and so on. i varies from 0 to 7.

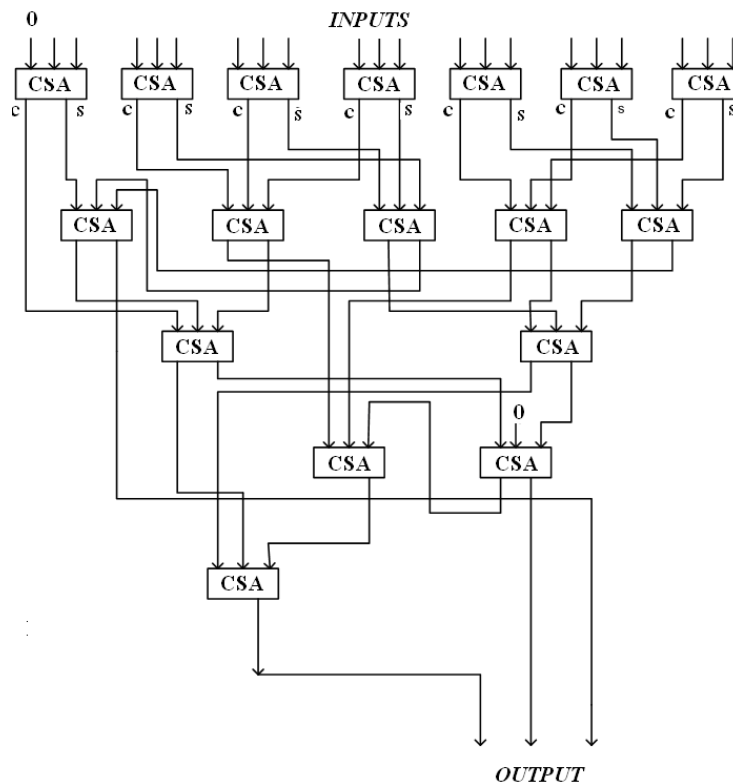


Figure 17. Carry Save Adder (CSA) tree when $p = 20$ and $N = 8$

When the contents of an s-LUT are large and cannot fit in a single memory module, then more than one memory modules or CAM-ROMs should be used. Figure 19 shows one s-LUT implemented using two CAM-ROM pairs. The CAM returns a zero ROM address for entries not present in it and as a result of this, output from all ROMs can be OR gated to get one valid result of that s-LUT.

Block 7: In this block gray level values of non-discarded templates are copied to templates that were discarded. The approach used is that, a discarded template is assigned the gray level value of a template that was not discarded and has the nearest highest number appended to it in Step 2 of the algorithm. The Boolean expressions representing the integrated circuit that performs operations in this block are shown below:

$$\begin{aligned}
 a_i &\leftarrow \overline{f_i(p)} \cdot \overline{f_i(p+1)} \cdot f_i(p+2), \text{ where } i=0 \text{ to } 7 \\
 a_8(0..7) &\leftarrow a_0 \cdot c_1(0..7) + a_1 \cdot c_2(0..7) + a_2 \cdot c_3(0..7) + a_3 \cdot c_4(0..7) + a_4 \cdot c_5(0..7) + a_5 \cdot c_6(0..7) + a_6 \cdot c_7(0..7) + a_7 \cdot c_8(0..7) \\
 a_9 &\leftarrow a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 \\
 a_{10}(0..7) &\leftarrow a_9 \cdot a_8(0..7) + \overline{a_9} \cdot b_8(0..7) \\
 gray_level_{t_0}(0..7) &\leftarrow a_{10}(0..7)
 \end{aligned} \tag{6.7}$$

where $gray_level_{t_0}$ is the gray level value corresponding to template t_0 .

$$\begin{aligned}
 b_i &\leftarrow \overline{f_i(p)} \cdot f_i(p+1) \cdot \overline{f_i(p+2)} \text{ where } i=0 \text{ to } 7 \\
 b_8(0..7) &\leftarrow b_0 \cdot c_1(0..7) + b_1 \cdot c_2(0..7) + b_2 \cdot c_3(0..7) + b_3 \cdot c_4(0..7) + b_4 \cdot c_5(0..7) + b_5 \cdot c_6(0..7) + b_6 \cdot c_7(0..7) + b_7 \cdot c_8(0..7) \\
 b_9 &\leftarrow b_0 + b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7 \\
 b_{10}(0..7) &\leftarrow b_9 \cdot b_8(0..7) + \overline{b_9} \cdot d_8(0..7) \\
 gray_level_{t_1}(0..7) &\leftarrow b_{10}(0..7)
 \end{aligned} \tag{6.8}$$

where $gray_level_{t_1}$ is the gray level value corresponding to template t_1 .

$$\begin{aligned}
 d_i &\leftarrow \overline{f_i(p)} \cdot f_i(p+1) \cdot \overline{f_i(p+2)}, \text{ where } i=0 \text{ to } 7 \\
 d_8(0..7) &\leftarrow a_0 \cdot c_1(0..7) + a_1 \cdot c_2(0..7) + a_2 \cdot c_3(0..7) + a_3 \cdot c_4(0..7) + a_4 \cdot c_5(0..7) + a_5 \cdot c_6(0..7) + a_6 \cdot c_7(0..7) + a_7 \cdot c_8(0..7) \\
 d_9 &\leftarrow d_0 + d_1 + d_2 + d_3 + d_4 + d_5 + d_6 + d_7 \\
 d_{10}(0..7) &\leftarrow d_9 \cdot d_8(0..7) + \overline{d_9} \cdot e_8(0..7) \\
 gray_level_{t_2}(0..7) &\leftarrow d_{10}(0..7)
 \end{aligned} \tag{6.9}$$

Where $gray_level_{t_2}$ is the gray level value corresponding to template t_2 .

$$\begin{aligned}
 e_i &\leftarrow \overline{f_i(p)} \cdot \overline{f_i(p+1)} \cdot \overline{f_i(p+2)}, \text{ where } i=0 \text{ to } 7 \\
 e_8(0..7) &\leftarrow e_0 \cdot c_1(0..7) + e_1 \cdot c_2(0..7) + e_2 \cdot c_3(0..7) + e_3 \cdot c_4(0..7) + e_4 \cdot c_5(0..7) + e_5 \cdot c_6(0..7) + e_6 \cdot c_7(0..7) + e_7 \cdot c_8(0..7) \\
 gray_level_{t_3}(0..7) &\leftarrow e_8(0..7)
 \end{aligned} \tag{6.10}$$

where $gray_level_{t_3}$ is the gray level value corresponding to template t_3 .

The four gray level values: $gray_level_{t_0}$, $gray_level_{t_1}$, $gray_level_{t_2}$ and $gray_level_{t_3}$ are stored at correct (row, column) coordinates in the output gray level image. The algorithm is pipelined in which each step can work independently on different inputs.

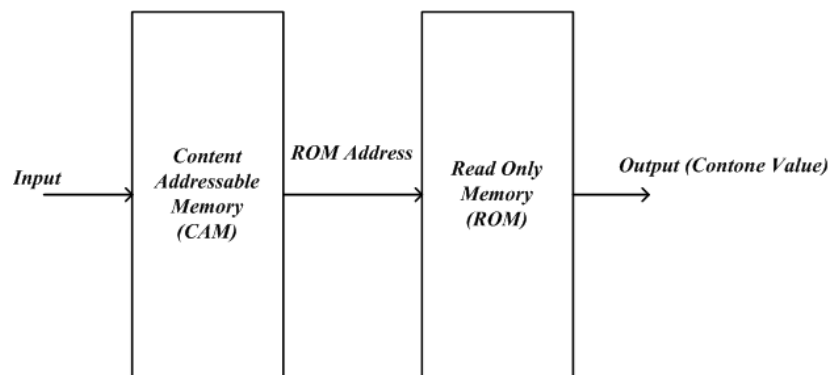


Figure 18. Smaller Look-Up Table (s-LUT) implemented in terms of CAM and ROM

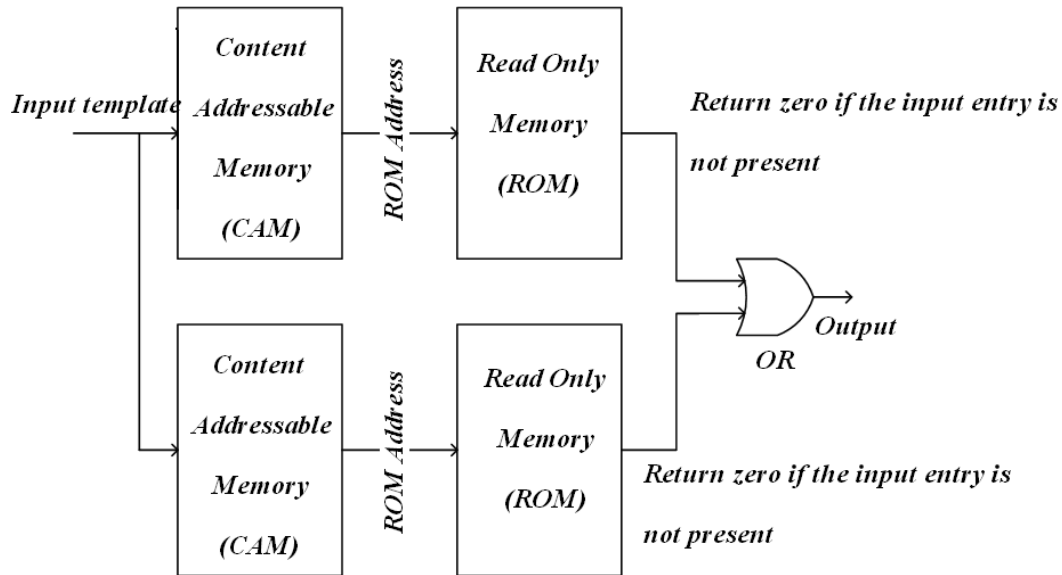


Figure 19. One s-LUT implemented using two ROMs and CAMs

7. HARDWARE IMPLEMENTATION

The computational part of the proposed algorithm for $k = 4$ and $N = 8$ is modeled using VHDL language and implemented on Altera Complex Programmable Devices (CPLD). It requires two CPLDs and external CAMs and SRAMs are used to store s-LUTs. Figure 20 illustrates the system block diagram. The CPLDs used are Altera [9] MAX II and CAM and SRAM are implemented in Altera APEX FPGA devices but can be replaced with discrete devices in future designs. CPLD I contains the proposed parallelization algorithm and CPLD II contains the pixel compensation circuit. The assignment of template numbers to incoming “19pels” is performed partially in both CPLD I & II in order to fit the design within MAX II pin count and to reduce fitting complexity of CPLD I.

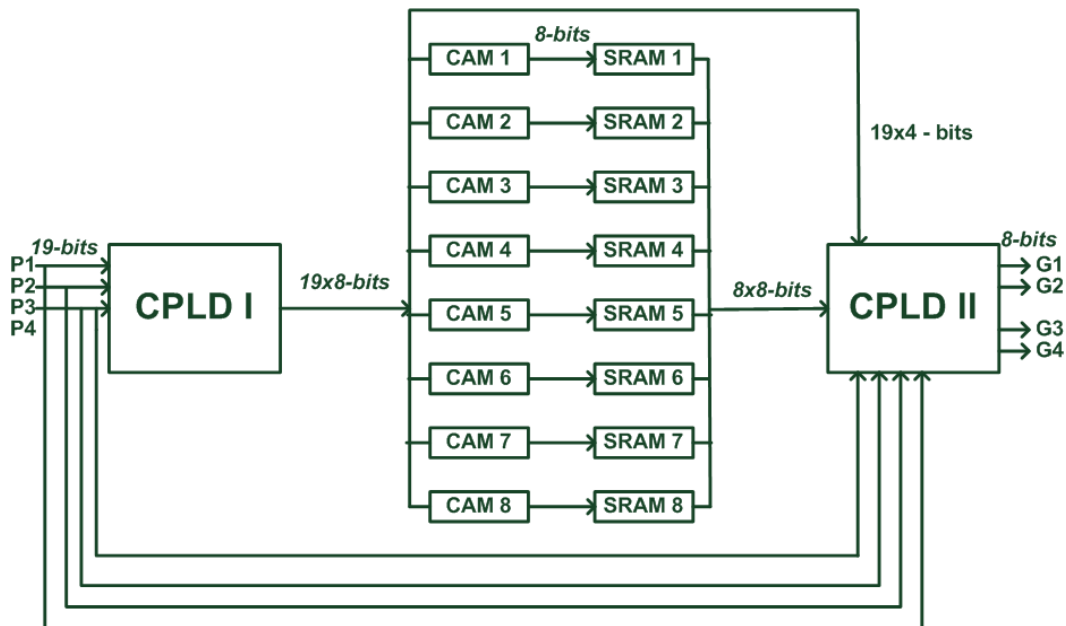


Figure 20. Block diagram of the algorithm implementation

In Figure 20, CPLD I accepts 4 “19pels” from the halftone image and sends each “19pels” according to the value returned by the XM function to its four outputs out of a total of eight output ports. The ports from CPLD I are connected to CAMs that are connected to SRAMs. The grey level values from SRAMs go to CPLD II where circuits for gray level value copying are present. The CPLD II gives grey level values in the correct sequence, *i.e.*, G1 corresponds to contone value of P_1 and so on. The results of CPLD implementation obtained from Fitter and Timing analyzer tools present in Altera Quartus II 5.0 are tabulated in Table 1.

Table 1. Results of CPLD Implementation

Device	Area	I/O pins	Clock Frequency
CPLD I EPM2210GF324I5	Logic elements: 2049/2210	261/272	33.86 MHz
CPLD II EPM2210GF324I5	Logic elements: 262/2210	262/272	164.85 MHz

8. CONCLUSIONS

The Look-Up Table (LUT) inverse halftone operation is parallelized by partitioning the LUT using the proposed algorithm. The LUT is partitioned into N smaller Look-Up Tables (s-LUTs) and, using them, up to k ($k \leq N$) pixels can be inverse halftoned simultaneously. The proposed algorithm integrates with the LUT method to perform parallel inverse halftoning. It has the following advantages: (a) instead of one pixel, now up to k pixels can be fetched and inverse halftoned simultaneously; (b) N smaller Look-Up Tables (s-LUTs) have total entries equal to the entries present in the single LUT of the serial LUT method; and (c) the image quality after parallelization remains comparable to the serial method. The hardware implementation of the proposed algorithm can be done using FPGA (Field Programmable Gate Array) or CPLD (Complex Programmable Gate Array) devices. The s-LUTs can be stored in separate CAMs and ROMs of sizes that match the sizes of the s-LUTs.

ACKNOWLEDGMENT

The authors acknowledge King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia for all support.

REFERENCES

- [1] Murat Mese and P. P. Vaidyanathan, “Recent Advances in Digital Halftoning and Inverse Halftoning Method,” *IEEE Trans. Circuits and Systems I*, June 2002, pp. 790–805.
- [2] Ping Wah Wong and Nasir D. Memon, “Image Processing for Halftoning,” *IEEE Signal Processing Magazine*, **20**(2003), pp. 59–70.
- [3] Murat Mese and P. P. Vaidyanathan, “Lookup Table (LUT) Method for Inverse Halftoning,” *IEEE Transactions on Image Processing*, **10**(2001), pp. 1566–1576.
- [4] A. N. Netravali and E. G. Bowen, “Display of Dithered Images,” *Proc. SID*, **22**(1981), pp. 185–190.
- [5] M. Y. Ting and E. A. Riskin, “Error-Diffused Image Compression Using a Binary to Gray Scale Decoder and Predictive Pruned Tree Structured Vector Quantization”, *IEEE Trans. Image Proc.*, **3**(1994), pp. 854–858.
- [6] P. C. Chang, C. S. Yu, and T. H. Lee, “Hybrid LMS–MMSE Inverse Halftoning Technique”, *IEEE Transactions on Image Processing*, **10**(2001), pp. 95–103.
- [7] Kuo-Liang Chung and Shih-Tung Wu, “Inverse Halftoning Algorithm Using Edge-Based Lookup Table Approach”, *IEEE Trans. Image Processing*, **14**(10)(2005), pp. 1583–1589.
- [8] R. Floyd and L. Steinberg, “An Adaptive Algorithm for Spatial Grey-scale”, *Proc. SID*, 1976, pp. 75–77.
- [9] <http://www.altera.com>