Algorithm for Parallel Inverse Halftoning using Partitioning of Look-Up Table (LUT)

Umair F. Siddiqi and Sadiq M. Sait
Department of Computer Engineering
King Fahd University of Petroleum & Minerals
Dhahran 31261, Saudi Arabia
Email: {umairf,sadiq}@kfupm.edu.sa

***Abstract*— The Look-Up Table (LUT) method for inverse halftoning is fast and computation-free technique employed to obtain good quality images. In this work we propose a new algorithm to parallelize the LUT method so that more pixels can be concurrently inverse halftoned using minimum additional hardware. The proposed algorithm partitions the single LUT of serial LUT method into N smaller Look-Up Tables (s-LUTs) such that the total number of entries in all s-LUTs remain equal to the number of entries in the single LUT of serial LUT method. The proposed algorithm can be implemented on a single FPGA (Field Programmable Gate Arrays) device with external memories to store s-LUTs.**

I. Introduction

The process of rendition of continuous tone pictures on media on which only two levels can be displayed is defined as Halftoning. It was first employed in the late 19^{th} century when printing machines were used to print images on paper. Halftoning was accomplished by adjusting the size of the dots according to local image intensity. This halftoning is called analog halftoning. With the proliferation of bi-level devices, digital halftoning also gained importance. Some of these bi-level devices are fax machines, printers, and plasma display panels. The input to a digital halftoning system is a continuous tone (or 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., 0 or 1. 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 [2]. LUT inverse halftoning was first introduced by Netravali and Bowen [3], but requires some information to be known that is not always available for halftone images. Subsequently Ting and Riskin [4] proposed another LUT method but it did not yield good image quality. In recent past a computation free LUT method was proposed by Mese and Vaidyanathan [2] which provides fast inverse halftoning with good image quality, and can be applied on several different halftones. Two more methods for LUT inverse halftoning [5], [6] were suggested by Kuo-Liang Chung et al. and P. C. Chang et al. They give better image quality but are not completely computation free. They require computation in addition to the Look-Up Table (LUT) access. They also have large LUT sizes as compared to the method of Mese et al. In 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 which is used to form the address for the LUT. Its precomputed 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 work, the LUT method of Mese and Vaidyanathan is referred to as serial LUT method. Recently, we proposed an algorithm [7] to perform parallel LUT inverse halftoning, where N smaller Look-Up Tables (s-LUTs) are generated using a training set that comprises halftone images and continuous tone images. The total number of entries in all N s-LUTs may become large and as a result of this the benefit of parallel inverse halftoning starts reducing because of the increase in memory requirement. In this paper, we present an algorithm that can perform parallel inverse halftone operation by partitioning the single LUT of serial LUT method into N number of smaller Look-Up Tables (s-LUTs). The N s-LUTs contain total entries equal to the entries in the single LUT of serial LUT method. In this way, the proposed algorithm can provide significant advantage in speed of inverse halftone operation and at the same time provides 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. The rest of this paper is organized as follows: First the serial LUT method is described then the parallelization of LUT method for inverse halftoning is described in detail that basically employs partitioning the LUT based on some observations. This is followed by the simulation of the proposed algorithm and discussion about its performance. In the last section, hardware modeling of the proposed algorithm for Field Programmable Gate Arrays (FPGA) devices is briefly discussed.

II. Look-Up Table (LUT) Method for Inverse Halftoning

In the LUT method that is also called serial LUT method templates are fetched from the halftone image following a

raster-scan style, i.e., a row is scanned from left to right and then the next row is scanned. 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 of halftone images and corresponding continuous tone images. The templates are fetched from 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 corresponds to that template in the training set. The inverse halftone operation is performed in this way that a template (t) is fetched from the halftone image and it 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) undergoes through anyone of these methods: (a) Low pass filtering, or (b) Best linear estimator [1]. The LUT method for inverse halftoning can also be applied to color halftones where separate LUTs exist for color planes R, G, and B.

III. Parallel Look-Up Table (LUT) Inverse Halftoning

In parallel LUT inverse halftoning, two or more templates are concurrently fetched from the halftone image and their contone values are obtained simultaneously. The problems associated in performing parallel LUT inverse halftoning are: (1) The LUT is a single memory block that does not allow accessing two or more locations simultaneously, and (2) The LUT method cannot be parallelized as it is because the memory requirements becomes very large due to the storage of a copy of LUT for each pixel to be inverse halftoned concurrently. The next section presents an algorithm to parallelize the LUT method for inverse halftoning while solving the above problems.

IV. Algorithm to Perform Parallel LUT Inverse Halftoning

This section shows the algorithm that can perform parallel inverse halftone operation by enhancing the serial LUT method. In the proposed algorithm N smaller Look-Up Tables (s-LUTs) are used in place of a single LUT. The proposed algorithm also introduces a functional block that returns unique numbers for the k templates that are fetched concurrently from the image. As a result of these two modifications, k templates can be fetched simultaneously and go through parallel inverse halftone operation using N s-LUTs. The parallel inverse halftoning using s-LUTs is accomplished by using two algorithms (1) Algorithm to generate smaller Look-Up Tables (s-LUTs), and (2) Algorithm to send k concurrently fetched templates to distinct s-LUTs. In the rest of this section the algorithms are described in detail.

A. Idea behind the Proposed Algorithm

The proposed algorithm is based on the idea to partition 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 assigned to each s-LUT. This approach has the problem that during inverse halftone operation it becomes difficult to find which template value exists in which s-LUT. The algorithm proposed in this paper, instead partition the LUT into N s-LUTs by using a new approach. The new approach is initiated after some halftone images are observed and it was found that adjacent, i.e., either top-bottom, or left-right template values differ from each other in terms of number of ones present in them. A function named as XOR with Mean (XM) is then defined. It takes XOR between the template that is fetched from the halftone image with 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 all concurrently fetched templates, i.e., k unique values are obtained. However, these values vary from 0 to $2^p - 1$, whereas number of s-LUTs is equal to N. In the next step, $mod\ N$ operation is applied and numbers in range from 0 to $N-1$ are obtained for all templates. The function is shown in Figure 1. Taking $mod\ N$ is computation free when N is a multiple of 2, i.e., 2, 4, 8, 16, etc and $mod\ N$ is performed by keeping only least significant $log_2 N$-bits of the input. The number of times XM function is able to return unique results for the templates is illustrated in graph in Figure 2. The y-axis shows the percentage of times XM function is able to return unique results for all templates that are fetched simultaneously and x-axis shows different values of N i.e, Number of s-LUTs. The different lines in the graph are drawn for different values of k. The N s-LUTs will be stored in N external memories and templates fetched from the halftone image act as input addresses to those memories. The distribution of templates among N s-LUTs may not be uniform and some s-LUTs require more than one memory blocks or blocks of large sizes as compared to the blocks of other s-LUTs. Two other approaches that can also be used in place of the XM function are: (1) Add the bits in the templates and then take $mod\ N$, and (2) Directly take $mod\ N$ of the fetched

Function XM($t_0(0 \cdots p-1), N, m$);
(*$t_0(0 \cdots p-1)$ is the input template having p-bits*)
(*N is the number of smaller Look-Up Tables (s-LUTs) and $N \in 2^i$ (where i= 1, 2, 3, etc)*)
(*m is the mean of all entries in s-LUTs*)
Begin
$r_0(0 \cdots p-1) = t_0(0 \cdots p-1)\ XOR\ m(0 \cdots p-1)$
$s_0(0 \cdots log_2 p-1) = r_0(0) + r_0(1) + \cdots + r_0(p-1)$
$y(0 \cdots log_2 N-1) = s_0(0 \cdots log_2 N-1)$
return $y(0 \cdots log_2 N-1)$
End

Fig. 1. Illustration of function XOR with Mean (XM).

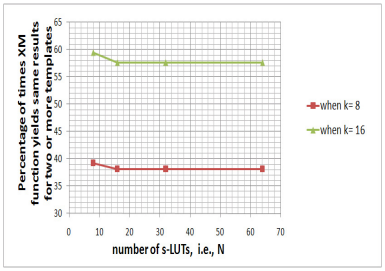

Fig. 2. Graph showing performance of XM function to return unique results for simultaneously fetched templates.

Algorithm Build_$s-LUTs(N)$
($*N$ is the number of smaller Look-Up Tables (s-LUTs)*)
Begin
1. Build 'Training_set' comprising of gray level images and corresponding halftone images
2. Number $s-LUTs$ from 0 to $N-1$
While ('Training_set' is not empty)
{
int index=0
t=Call Fetch_Template(index)
index++
r=Call XM(t)
Store t and its gray level value in $s-LUT$ having number equal to r }

Fig. 3. Illustration of the algorithm to generate $s-LUTs$.

template. However, from experimentation it was found that XM function yielded best image quality compared to others and therefore the algorithm proposed in this paper uses only this approach.

B. Algorithm to Generate N smaller Look-Up Tables ($s-LUTs$)

N number of smaller Look-Up Tables (s-LUTs) must be generated before inverse halftone operation is performed. The s-LUTs are numbered from 0 to $N-1$, where N must be a multiple of 2 i.e., 2, 4, 8, etc. The algorithm is shown in Figure 3. It starts by building a *'Training_set'*, that comprises gray level images and halftone images. The next step is to fetch all templates and their corresponding gray level values from the training set and store them in the s-LUTs according to the procedure shown in the algorithm in Figure 3. The same value of N chosen to generate s-LUTs should be used in the algorithm to perform inverse halftone operation, that is described in the next section.

C. Algorithm to Perform Parallel Inverse Halftoning using s-LUTs

The algorithm to perform parallel inverse halftoning is illustrated in Figure 4. The algorithm starts by concurrently fetching k templates from the halftone image. Then function XM is simultaneously applied to each template. In the next step, the results from XM functions are checked and if results

Algorithm Parallel_Inverse_Halftoning($Halftone\_image, N, k$);
($*Halfone\_image$ is the input halftone image*)
($*N$ is the number of smaller Look-Up Tables (s-LUTs)*)
($*k$ is the number of concurrent templates*)
Begin
for (i=0;i< image_height;i++)
for (j=0;j< image_width;j++) {
for (n=0; $< k$**; n++)**{
t_n=Call Fetch_Template(i,j+k) }
for (n=0; n< k**; n++)**{
r_n=Call XM(i,j+n) }
for (m=k;m>1;m–) {
sel=r_m
for (p=m;p>0;p–) {
if sel==r_p
Call DISCARD(r_p) } }
for (n=0; n< k**; n++)** {
Gray_Image(i,j+n)=Call $s-LUT_{r_n}(t_n)$ }
for (n=0; n< $k-1$**; n++)** {
if r_n was DISCARDED or its contone value does not exists in its $s-LUT$
r_n=r_{n+1} }
End

Fig. 4. Illustration of the algorithm proposed to perform parallel inverse halftoning using s-LUTs.

of any two or more templates are same then only one template among them in kept and the remaining templates that have the same result are discarded. In the next step, the non-discarded templates are sent to the s-LUTs that have same numbers as their results returned from the XM function. In the following step, the templates that were discarded or the templates that do not find their contone values in their respective s-LUT are assigned contone values by copying from their neighbors that already obtained their contone values from s-LUTs. The algorithm continuous until all pixels in the halftone image are inverse halftoned. This algorithm is pipelined in which each operation can be performed in parallel on different data inputs and new k templates can be fetched on every clock cycle from the halftone image. The clock cycles consumed in performing parallel LUT inverse halftoning using the proposed algorithm are equal to: 2 $\times$ (pipelines stages) + (number of pixels in the halftone image) $\div$ k. On the other hand, the serial LUT method requires clock cycles equal to the number of pixels in the halftone image.

V. Simulation

The proposed algorithm and the serial LUT method are implemented using Java programming language. The training set comprises 17 images. First N s-LUTs are generated and the partitioning of templates among s-LUTs is performed. The graph in Figure 5 shows partitioning when $N=8$. The test set that is used to perform inverse halftone operation consists of images: 'Boat', 'Clock', and 'Lena'. The test images are not included in the training set. The results are shown in Table I.

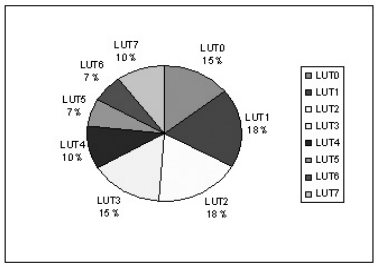

Fig. 5. Graph showing partitioning of templates to s-LUTs when $N = 8$.

TABLE I
SIMULATION RESULTS

Image	N	k	PSNR1	PSNR2
Boat	8	4	28.5448 dB	30.1861 dB
Clock	8	4	30.08466 dB	30.1680 dB
Peppers	8	4	29.2605 dB	29.4154 dB
Boat	16	4	28.5541 dB	30.1861 dB
Clock	16	4	30.0786 dB	30.1680 dB
Peppers	16	4	29.2199 dB	29.4154 dB

(PSNR1= PSNR of the image obtained from the proposed algorithm and PSNR2= PSNR of the image obtained from the serial LUT method)

The table shows the image quality in terms of Peak Signal to Noise Ratio (PSNR) of the images obtained from the proposed algorithm for given values of k and N. The table also shows the image quality of the images obtained from the serial LUT method. The proposed algorithm offers k times saving in LUT entries while concurrently inverse halftoning k pixels compared to the the serial LUT method if it is implemented multiple times to concurrently inverse halftone k pixels. The previous algorithm of the authors [7] was able to offer up-to 4 times saving, that also varies with the number of images present in the training set.

Fig. 6. Inverse halftoned image obtained from the proposed algorithm (PSNR= 27.6723 dB).

VI. HARDWARE MODELING

The computational portion of the proposed algorithm with values k= 4 and N= 8 is modeled in VHDL and synthesized for a single Xilinx Spartan 3E FPGA using Xilinx ISE tools. The results from Xilinx tools show that it consumed 958 out of 960 slices, maximum frequency reported is 101.68 MHz, and the equivalent gate count is found to be 14,729. The design assumes that s-LUTs will reside on external memories. The outputs from FPGA will become the addresses for external memories and the results from memories will come back into the FPGA in order to assign contone values to templates that are discarded or that do not find their contone values in their s-LUTs.

VII. CONCLUSION

A new algorithm to perform parallel inverse halftoning using s-LUTs is proposed. It offers significant speedup in inverse halftone operation over the serial LUT method. It partitions the contents of the single LUT of serial LUT method into N s-LUTs. The number of entries in N s-LUTs remains equal to the number of entries in the single LUT. The image quality is slightly less than the image quality of the serial LUT method. The algorithm is pipelined in which each operation can perform concurrently on outputs of its predecessor operation. It takes a single FPGA for implementing the computational portion of the algorithm and s-LUTs will reside on external memories.

ACKNOWLEDGMENT

Authors would like to thank 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.

[2] Murat Mese and P. P. Vaidyanathan, *"Look-Up Table (LUT) Method for Inverse Halftoning,"* IEEE Trans. Image Processing, vol. 10, October 2001.

[3] A. N. Netravali and E. G. Bowen, *"Display of Dithered Images,"* Proc. SID, vol. 22, pp. 185-190, 1981.

[4] 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 Proceeding, Vol. 3, pp. 854-858, 1994.

[5] P. C. Cheng, C. S. Yu and T. H. Lee, *"Hybrid LMS-MMSE Inverse Halftoning Technique,"* IEEE Trans. Image Processing, vol. 10, January 2001.

[6] Kuo-Liang Chung and Shih-Tung Wu, *"Inverse Halftoning Algorithm using Edge-Based Look-Up Table Approach,"* IEEE Trans. Image Processing, vol. 14, Issue 10, oct. 2005, pp. 1583-1589.

[7] Umair F. Siddiqi, Sadiq M. Sait, and Aamir A. Farooqui, *"Parallel Algorithm for Hardware Implementation of Inverse Halftoning,"* IEEE ISCAS 2005, Vol. 3 pp. 2377 - 2380.