A VLSI Progressive Coding for Wavelet-based Image Compression

Tsung-Hsi Chiang and Lan-Rong Dung, *Member*, IEEE

Abstract — *This paper presents a new algorithm for progressive image coding, called Tag Setting In Hierarchical Tree (TSIHT). The TSIHT coding can save the memory requirement while keeping the low-bit-rate quality high. The TSIHT algorithm has been implemented onto a chip with 0.35μ 1P4M CMOS technology. The chip can handle 256×256 gray-scale images and the gate count is about 2560 gates within 247500 μm^2 area. The latency of the critical path is 6.32 ns, and the maximum working frequency can be as high as 158 MHz.*[1]

Index Terms — Image compression; VLSI; Progressive coding

I. INTRODUCTION

Rapidly growing number of high-resolution images have come with the advancement of consumer products in various multimedia applications. Due to the huge amount of data involved, even a compressed image is significant in size, large image sent over low-bandwidth links will still need lengthy transmission-time, especially in computation-limited or network-limited environment, such as the portable device. A better solution is to encode image as transmit bit-stream simultaneously and progressively. Progressive image transmission (PIT) technique provides the capability that it allows to interrupt the transmission when the quality of the received image has reached a desired accuracy. When the receiver recognizes that the image is not interesting or only a specific portion of the complete image is needed, PIT can also terminate transmission at any point of bit-stream. Newer coding techniques, such as JPEG2000 [1]-[2], and MPEG4 [3] standards, have supported the progressive transmission feature.

PIT via wavelet-coding using the Embedded Zerotree Wavelet (EZW) algorithm was firstly presented by Shapiro [4] in 1993. Later in 1996, Said and Pearlman [5] presented a better implementation based on Set-Partitioning In Hierarchical Trees (SPIHT) underlying the principles of EZW method. The set-partitioning algorithm uses the principles including self-similarity across scales as in EZW, partial ordering by magnitude of the wavelet-coefficients, set-partitioning into hierarchical tree, and ordered bit-plane transmission of the refinement-bits. Due to the excellent performance in peak-signal to noise ratio (PSNR) [5] measurement, SPIHT coder has been approved that the encoding procedures are faster and more efficient than EZW coding. Therefore, many coding algorithms have been developed by [6]-[10] based on SPIHT coding.

However, the memory requirements of the SPIHT-based algorithms are incongruous for hardware design. Considering SPIHT coding in a practical implementation, the significance information of image is stored in three ordered lists, called list of insignificant sets (LIS), list of insignificant pixels (LIP) and list of significant pixels (LSP). For a typical 256×256 gray-scale image, each entry of the lists requires at least 8+8=16 bits to store the row and column coordinate values. Thus, SPIHT coding needs 2×16×256×256 bits = 256 *K* bytes memory to store both LIP and LSP lists. In addition to LIS list, it also needs 2×16×256×256 bits = 256 *K* bytes memory, where each entry requires 2 bits to indicate type A or type B node of LIS list. Totally, SPIHT coding needs 512 *K* bytes memory for a 256×256 gray-scale image, in worse case.

In this paper, we suggest a new coding algorithm for progressive image transmission called Tag Setting In Hierarchical Tree (TSIHT). The proposed TSIHT coding keeps low bit-rate quality as SPIHT algorithm and has three improved features. Firstly, to reduce the amount of memory usage, TSIHT coding introduces tag flags to store the significant information instead of the coordinate-lists in SPIHT. The tag flags are four two-dimensional binary tag-arrays including Tag of Significant Pixels (TSP), Tag of Insignificant Pixels (TIP) and Tag of Significant Trees (TST) respectively. When comparing with SPIHT coding, TSIHT only needs 26 *K* bytes memory to store four tag-arrays for a 256×256 gray-scale image. Secondly, both sorting-pass and refinement-pass of SPIHT coding are merged in one in TSIHT coding in order to simplify hardware-control and save unnecessary memory. Finally, TSIHT uses the Depth-First-Search (DFS) traversal order to encode bit-stream rather than the Breadth-First-Search (BFS) method as the SPIHT coding. Since, DFS method searches the root node and each one of the branching node of the immediate descendants until it reaches the deepest leaves. For the hierarchical pyramid nature of the spatial orientation tree, DFS provides a better architecture than BFS method.

Additionally, a VLSI image compressor called PIE (Progressive Image Encoder) core for TSIHT coding has been implemented onto a chip with 0.38 μm one-poly-four-metal CMOS technology. The prototype of PIE core can handle 256×256 gray-scale images. The gate count of PIE core is about 2560 gates within 247500 μm^2 area. The latency of the critical path is 6.32 ns, and the maximum working frequency is about 158 *MHz*.

The remainder sections of this paper are organized as follows. Section 2 is the background of progressive image

[1] This work was supported in part by the National Science Council, Taiwan, under the grant number NSC 95-2221-E009-337-MY3 and Chung-Shan Institute of Science and Technology, Taiwan, under the project BV94G10P.

Tsung-Hsi Chiang and Lan-Rong Dung are with the Departmetn of Electrical and Control Engineering, National Chiao Tung University, Taiwan, R.O.C. (aries.chiang@msa.hinet.net and lennon@faculty.nctu.edu.tw)

Contributed Paper
Manuscript received January 12, 2007

transmission, EZW and SPIHT coding. Section 3 addresses the proposed TSIHT coding algorithm. Section 4 addresses the software implementation of TSIHT coding. Section 5 presents the VLSI architecture of the proposed PIE core. Section 6 shows the synthesis result and performance analysis of PIE core. Finally, the conclusion is given in Section 7.

II. BACKGROUND

A. Progressive Image Transmission

A key for the progressive image transmission is to apply multi-resolution decomposition on the target image. The multi-resolution decomposition provides multi-resolution representation of an image. At different resolution, the details of an image characterize different physical structures of the scene. At a coarse resolution, these details correspond to the larger structures which provide the image content. It is therefore natural to analyze the image details at a coarse resolution first and then gradually increase the resolution. Usually, multi-resolution decomposition defines a set of orthonormal basis, such as the Haar basis [11] in wavelet-decomposition. By applying decomposition-transformation, it is possible to represent an image based on the coefficients in an orthonormal basis expansion.

Let $p_{i,j}$ be a two-dimensional image, where i and j are the indices of pixel coordinates. The multi-resolution decomposition of image $p_{i,j}$ is written as

$$c = \Omega(p). \tag{1}$$

Where $\Omega(\cdot)$ is a transformation of multi-resolution decomposition. Two-dimensional coefficient array c has the same dimensions as image p, and each element $c_{i,j}$ is the transformation coefficient of p at coordinate (i,j). In a progressive image transmission, receiver updates received reconstruction coefficient c_r according to the coded message until approximate or exact amount coefficients have been received. Then, the decoder can obtain a reconstructed image by applying inverse transformation

$$p_r = \Omega^{I}(c_r). \tag{2}$$

Where pr is the reconstructed image, and c_r are progressively received coefficients. Image distortion of reconstructed image pr from original image p can be measured by using Mean Squared Error (MSE), that is

$$D_{MSE}(p - p_r) = D_{MSE}(c - c_r) = \frac{1}{MN}\sum_i \sum_j (c_{i,j} - c_{r\,i,j})^2. \tag{3}$$

Where MN is the total number of all image pixels. In a progressive image transmission process, the transmitter rearranges the details within the image in the decreasing order of the importance. From Equation (3), it is clear that if an exact value of the transform coefficient $c_{ri,j}$ is sent to the decoder, then the MSE decreases by $|c_{i,j}|^2 / MN$ [5].This means that the coefficients with larger magnitude should be transmitted first because they have a larger content information.

B. Embedded Zerotree Wavelet Algorithm

Embedded Zerotree Wavelet (EZW) algorithm was suggested by Shapiro in 1993 [4]. Shapiro uses a simple and general model to describe the distribution of the wavelet coefficients obtained from a Discrete Wavelet Transformed (DWT) image. The DWT decomposes the input image into a set of sub-bands in multiple resolutions. Generally, most of the energy in an image is concentrated in the low-frequency region. When an image is subjected to n-level decomposition using DWT, the n-th level would correspond to the lowest frequency sub-band and would correspond to the coarsest resolution. Thus, when one moves from higher levels to lower levels of sub-band decomposition, there would be a decrease in the energy content of the sub-band. In each decomposition level, the coarse sub-band is a low-pass approximation of the original image, and the other sub-bands are finer scale refinement. Every coefficient in each decomposition level, expect the highest frequency, can be related to a set of coefficients of similar orientation at the next finer scale level. The coefficient at the coarse scale is called the parent, and all coefficients at the same spatial location of similar orientation at the next finer scale level are called the children.

To rearrange the wavelet coefficients in the decreasing order of the importance, Shapiro defines the *insignificant* concept and a data structure *zerotree*. A wavelet coefficient x is said to be insignificant with respect to a given threshold T if $|x|<T$. The zerotree is a tree structure based on the hypothesis that if a wavelet coefficient at a coarse scale is insignificant with respect to a given threshold T, then all wavelet coefficients of the same orientation in the same spatial location at finer scales are also significant with respect to T. To coding wavelet coefficients, EZW algorithm uses zigzag-scanning coding sub-band by sub-band. The scanning order of wavelet coefficients begins from the lowest frequency sub-band and continuously searches with the breadth-first-search (BFS) to the higher frequency sub-bands. Parents are scanned before any of their children, but after all neighboring parents have been scanned. Each coefficient is compared against the current threshold T. A coefficient is significant if its amplitude is greater than T. A significant coefficient is then encoded using one of the symbols NS (negative significant) or PS (positive significant). The ZTR (zerotree root) symbol is used to signify an insignificant coefficient. The IZ (isolated zero) symbol signifies a coefficient below T but with at least one child not below T. For significant coefficients, EZW further encodes coefficient values using a successive approximation quantization (SAQ) scheme. SAQ uses to provide a multi-resolution representation of the coefficients and to facilitate the embedded coding. Then, coding is done bit-plane by bit-plane.

C. Set Partitioning in Hierarchical Tree

Set Partitioning In Hierarchical Tree (SPIHT) is an image coding algorithm suggested by Said and Pearlman in 1996 [5]. Based on EZW [4] concept framework, SPIHT algorithm provides a better performance and less complexity

implementation than EZW. The EZW coding is essential to compress the ordering information as conveyed by the results of the significance tests. In SPIHT algorithm, the ordering data is not explicitly transmitted. Instead, SPIHT coding algorithm uses a partitioning of trees in a manner that it tends to keep insignificant coefficients together in large subsets. Herein, the subset partitioning is so effective, and the significance information is so compact that SPIHT has better performance than the EZW algorithm.

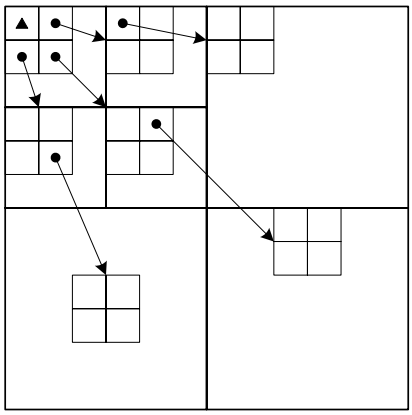


Fig. 1. Parent-offspring dependencies in a spatial orientation tree

The SPIHT algorithm exploits the coefficients obtained from DWT transformed image. These DWT pyramid coefficients are energy compaction, cross sub-band similarity and decaying of coefficient magnitude across sub-bands. To apply SPIHT algorithm, a tree structure, called *spatial orientation tree*, is defined as the spatial relationship on the hierarchical pyramid. The tree is defined in such a way that each node has either no offspring (the leave) or four offsprings, which form a group of 2×2 adjacent pixels. For instance, Figure 1 indicates the spatial orientation tree and corresponding parent-offspring relationships across the sub-bands. The pixels in the highest level of the pyramid are tree roots, and 2×2 adjacent pixels are grouped into blocks. Their branching offsprings, except the leaves, also have 2×2 adjacent pixels grouped into blocks. Let $c_{i,j}$ denote the wavelet transform coefficient at *(i,j)* in the DWT transformed image. The parent-offspring linkage, except at the highest and the lowest pyramid levels, is

$$O(i,j)=\{(2i,2j),(2i,2j+1),(2i+1,2j),(2i+1,2j+1)\}. \quad (4)$$

Where *O(i,j)* is the set of all immediate descendants of node *(i,j)*. The set of all descendants of the node *(i,j)* is denoted by *D(i,j)*. The set of all descendants *D(i,j)* but excluding immediate descendant is *L(i,j)=D(i,j)* \ *O(i,j)*. The relationships between these coordinate sets are shown in Figure 2. While performing the SPIHT coding algorithm, it needs three ordered lists including list of insignificant set (LIS), list of insignificant pixels (LIP), and list of significant pixels (LSP) to store the signification information. In LIP and LSP, the entries represent individual pixels identified by coordinate *(i,j)*. In LIS list, each entry represents an element either in the set *D(i,j)* with type A or in the set *L(i,j)* with type B.

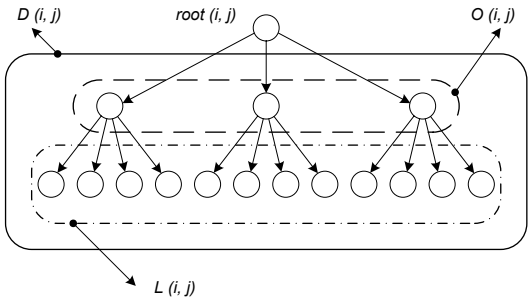


Fig. 2. Relationship between different coordinate sets

The essential of SPIHT coding algorithm is to identify which coefficients are significant, sort selected coefficients in each *sorting pass*, and transmit the ordered refinement bits. A function $S_n(T)$ is used to indicate the significance of a set of coordinates *T*, that is

$$S_n(\mathrm{T})=\begin{cases}1, & \max_{(i,j)\in T}\{|c_{i,j}|\}\geq 2^n,\\ 0, & \text{otherwise.}\end{cases} \quad (5)$$

During the *sorting pass*, the pixels in the LIP, which was insignificant in the previous pass, are tested, and those that become significant are moved to the LSP. Similarly, it tests each entry in LIS list. When a set if found to be significant, it is removed from the list and partitioned. The new subsets with more than one element are added back to the LIS, while the single-coordinate set are added to the end of the LIP or the LSP. The LSP contains the coordinates of the pixels that are visited in the *refinement pass*. Then, for each entry *(i,j)* in the LSP, except those included in the last *sorting pass*, output the *n*-th most significant bit of $|c_{i,j}|$. Node tests and descendant tests are performed with the maximum threshold 2^N first. Then, it repeats with smaller threshold, 2^n, n=N-1, N-2,…, iteratively until the compressed bit amount reaches a predefined value.

III. PROPOSED TSIHT CODING ALGORITHM

A. Principles

The proposed TSIHT (Tag Setting in Hierarchical Tree) coding is based on SPIHT algorithm. To implement the TSIHT on a silicon, we examine the performance and memory requirement of a hardware implementation. In our opinion, the TSIHT coding has three essential advantages as following.

(1) Less memory required:

When applying SPIHT algorithm, large amount of memory may be occupied to store LSP, LIP and LIS lists. Instead, the TSIHT coding algorithm uses three *tag* flags including TSP (Tag of Significant Pixels), TIP (Tag of Insignificant Pixels) and TST (Tag of Significant Tree) to distinct different entries in LSP, LIP and LIS

respectively. For a typical 256×256 gray-scale image, both of each TSP and TIP lists need 256×256 bits, and TST list needs 128×128 bits. Thus, TSIHT coding totally needs 2×256×256+128×128 (bits) = 18 *K* bytes memory. It is apparent that the proposed TSIHT coding occupies less memory than SPIHT, which needs 2×16×256×256 (bits) = 250 *K* bytes to store three lists.

(2) Improved *refinement pass*

In the SPIHT algorithm, *refinement pass* is to output the *n*-th most significant bit of $|c_{i,j}|$ which is in the LSP list except those included in the last *sorting pass*. To implement coding algorithm on a chip, SPIHT needs more hardware control and more memory space to store extra information in *refinement pass*. However, there is no precedence relation between the *sorting pass* and the *refinement pass*. A better approach proposed in TSIHT is to put *refinement pass* before the *sorting pass*. Thus, TSIHT does not need to store last address or information of the *refinement pass* and is more efficient than SPIHT coding.

(3) Efficient depth-first-search (DFS)

The spatial orientation tree is defined on the hierarchical pyramid. In the *sorting pass*, SPIHT coding algorithm uses breadth-first-search (BFS) to traverse all the nodes in the tree structure. In order to access each node in the tree, an input buffer is needed to hold all the ancestor-descendant relations of the coefficients. Thus, the location addresses of the four immediate descendants of a node can be calculated systematically. As showing in Figure 3, it is easily to calculate the addresses from root node to the descendants within the first three steps. However, while proceeding to step 4, there is no more ancestor-descendant relations to calculate node addresses. In proposed TSIHT algorithm, we use depth-first-search (DFS) [12] method instead. As showing in Figure 4, the DFS method searches the root node and each one of the branching to the immediate descendants until it reaches the leaves. By using DFS method, the address generation of the ancestor-descendant coefficients is more efficient than SPIHT coding.

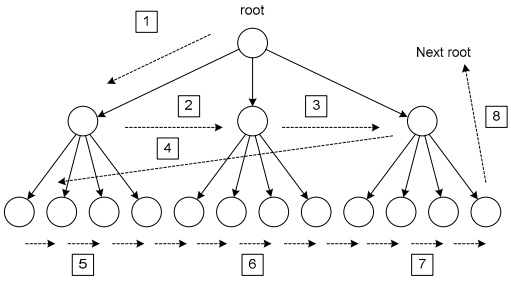


Fig. 3. BFS traversal in SPIHT coding

Proposed TSIHT coding is based on SPIHT algorithm extended by using above principles. Besides, in our experimental result, it shows that TSIHT also keeps low bit rate quality as SPIHT does, even better. In the following section, we will discuss the implementation detail of TSIHT coding.

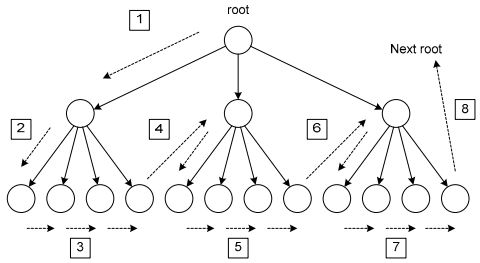


Fig. 4. DFS traversal in TSIHT coding

IV. SOFTWARE IMPLEMENTATION

Let TSP, TIP and TST be the two-dimensional binary arrays, whose entries are either '0' or '1'. The overall TSIHT coding algorithm includes six steps as follows.

(1) **Initialization**: output $n = \lfloor \log_2(\max\{|c_{i,j}|\}) \rfloor$; set each value of all entries in TSP, TIP and TST arrays to '0'.

(2) Refinement output:

(a) for each entry *(i,j)* in the TSP do:

(i) if TSP=1 then output the *n*-th most significant bit of $|c_{i,j}|$;

(3) TIP testing:

(a) for each entry *(i,j)* in the TIP do:

(i) if TIP=1 and $S_n(c_{i,j}) = 1$ then

(A) output '1' and output sign of $c_{i,j}$;

(B) set value TIP := 0 and TSP := 1;

(ii) otherwise, if TIP=1 and $S_n(c_{i,j}) = 0$ then output '0';

(4) TST update:

(a) for each entry $(k,l) \in O(i,j)$ do:

(i) if TST=0 and $S_n(c_{i,j}) = 1$ then set value TST:=1;

(5) Spatial orientation tree encoding:

(a) for each entry *(i,j)* using DFS method do:

(i) if TSP=0 and TIP=0 then

(A) if $S_n(i,j) = 1$ then output '1', sign of $c_{i,j}$ and the value of TST; set value TSP:=1;

(B) otherwise, if $S_n(i,j) = 0$ then output '0' and the value of TST; set value TIP:=1;

(6) Quantization-step update: decrease n by 1 and go to Step 2.

In Step 1, TSIHT coding first calculates initial threshold and sets the values of three tag flags TSP, TIP and TST to '0' initially. In Step 2, the entry marked with TSP=1, which is evaluated in the last Step 5, is significant. The entry, TIP=1, tested as insignificant in last Step 5 may be significant in Step 3 due to the different threshold. Thus, the algorithm performs TIP testing to update TIP value in Step 3. In Step 4, it updates TST value of each coefficient except the leave nodes and prepares to perform tree encoding in next Step. If a node is TST=0, its descendants are all insignificant; in the other words, the tree leading by that node, TST=0, is a zerotree. The algorithm searches those nodes, TST=0, using depth-first-search (DFS) method and outputs an '0' in Step 5 to keep low

bit rate as SPIHT coding does. At last, it decreases quantization step n by 1 and go to Step 2 iteratively.

Proposed TSIHT coding algorithm is the same as what the SPIHT coding does but using different data structures. For instance, in the *refinement output* and *TIP testing* steps, TSIHT uses tag flags TSP and TIP to indicate whether a node is significant or not. Then, TSIHT can output and encode the image stream by investigating the TSP and TIP tags. On the other hand, SPIHT coding uses coordinate sets LSP and LIP to store coordinate information of nodes. When comparing both methods, the information stored in TSP (LSP) is the same as in TIP (LIP). Besides, in the *spatial orientation tree encoding* step of TSIHT coding, if a node is TST=1, it trends to searching its descendants using DFS method without any output. However, in the *sorting pass* of TSIHT coding, each node in LIS list with type A may change to type B and apply encoding again. Thus, in general case, TSIHT has lower bit rate quality than SPIHT does.

To see the experimental result, we use Matlab as the software implementation to evaluate TSIHT coding algorithm. The relation of bit-rate and PSNR illustrated in Figure 5 is obtained by apply TSIHT coding on the 256×256 gray-scale lena image.

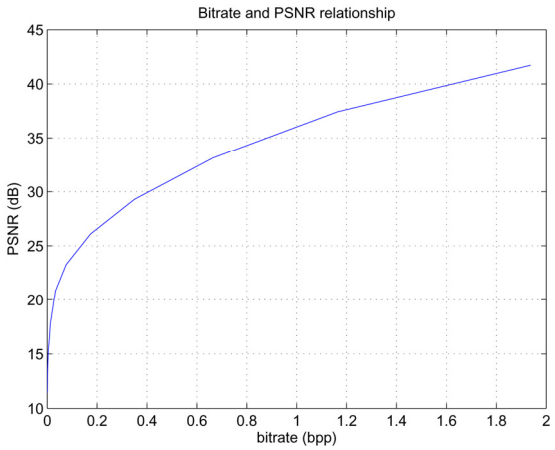


Fig. 5. Bit-rate and PSNR relationship of TSIHT coding

V. VLSI ARCHITECTURE OF TSIHT ENCODER

A. Architecture Overview

Based on the proposed TSIHT coding algorithm, a hardware implementation, called the Progressive Image Encoder (PIE), is introduced in this section. In our work, PIE is designed as a VLSI IP (Intellectual Property) core for the purpose of various image compression applications. The pin assignment for PIE IP core is shown in Figure 6. Note that, PIE reads the wavelet coefficients from external memory using a 16-bit input signal, **Coeff**[0:15], and it reads the tag flags of TSP, TIP and TST from external tag memory using 8-bit input signals, **TSP**[7:0], **TIP**[7:0] and **TST**[7:0] respectively. If PIE want to read coefficients or tags from memory, it first generates the address, **Addr**[15:0], of the data, and then it reads the data using input signals. PE outputs the encoded bit-stream using signal **bit_out** when sync asserts. The signals **TSP_in**, **TIP_in** and **TST_in** are used to output tag data. PIE uses read-enable signals, **TSP_ren**, **TIP_ren** and **TST_ren**, or write-enable signals, **TSP_wen**, **TIP_wen** and **TST_wen**, to control the reading or writing action of tag memory.

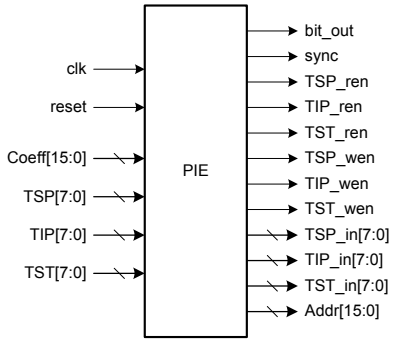


Fig. 6. Pin assignment for PIE IP core

In Figure 7, it shows the overall architecture of PIE encoder. Except the external coefficient and tag memory, PIE includes six blocks as following. *Address Generator*, which is the most complex component in PIE, generates the location addresses of the coefficient and the tag memory. Clock Divider generates three clock signals with different frequencies to synchronize internal circuit. *Threshold Generator* calculates the initial value *n* and updates its value at every iteration. Tag Access Unit controls the access of three tags, TSP, TIP and TST. Bit-Stream Generator outputs the encoded bit-stream of PIE. Controller is the master of all blocks. We will discuss each block in the following sections.

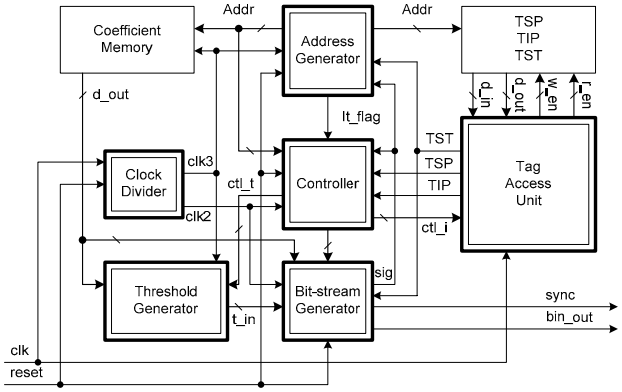


Fig. 7. Progressive Image Encoder hardware architecture

B. The Components of PIE

(1) *Address Generator* (AG): In order to access the coefficient and tag from external memory, Address Generator (AG) provides a mapping from the (*row,col*) coordinate to the linear address of memory. On the other words, AG is used to generate a 16-bit address signal, while the signal **Addr[15:8]** is the row address, and the signal **Addr[7:0]** is the column address, such that, each address pair to the coordinate of the coefficient or the tag can be located from memory. The hardware architecture of PIE is illustrated in Figure 8.

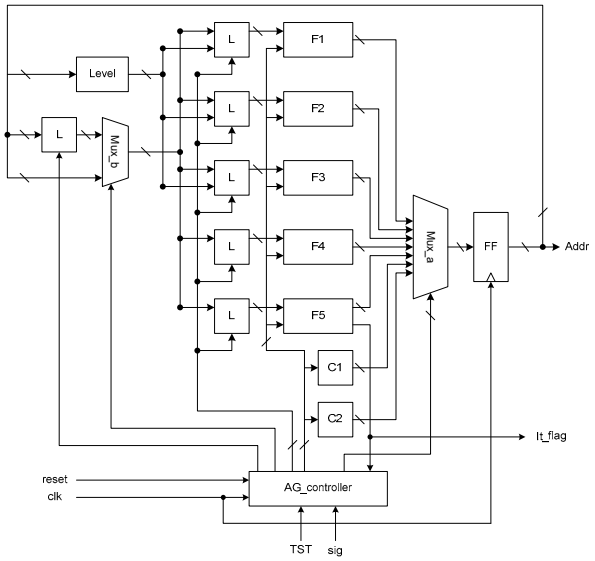


Fig. 8. Address Generator architecture

To adapt different data structures of external memory content, AG behaves as a mapping function from current address to the next address depends on five selection cases from F1 to F5 as following.

(A) F1: Wavelet coefficient address generation

When PIE performs TSIHT coding in *TST update* step, AG is used to generate the wavelet coefficient address with bottom-up direction. For an instance, the wavelet coefficients of third order DWT transform are shown in Figure 9. While updating TST, AG first searches the most peripheral starting at the start mark toward the inner nodes of every scanning line. Let *c_col* and *c_row* be the current column and row addresses; *n_col* and *n_row* be the next column and row addresses. Assuming *tmp_size* is the coordinate boundary in each level. The flowchart of the F1 address generation is illustrated in Figure 10. Note that, as showing in Figure 10, next address is obtained from current address depends on different boundary conditions.

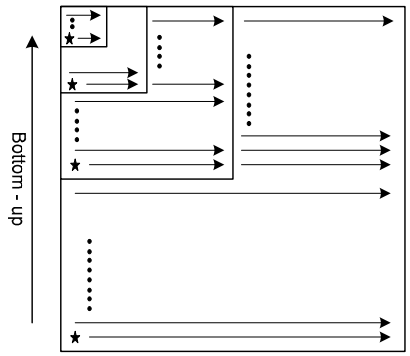


Fig. 9. Bottom-up searching direction

(B) F2: Ancestor address generation

In *TST update* step, for each entry *(k,l)*$\in$*O(i,j)*, if it finds that a descendant coefficient, *(k,l)*, with TST=0 is significant, the TST value of the parent, *(i,j)*, assigned to TST=1. To locate the ancestor address from its descendant coefficient, bitwise-shifting operation on descendant coordinate is used. For instance, Figure 11 illustrates the ancestor-descendant relations labeled with row and column address. The ancestor address can be obtained by right-shifting one bit on each of its descendant coordinate.

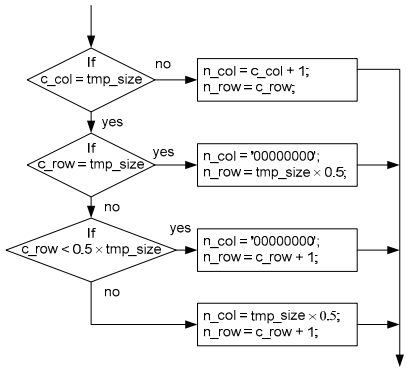
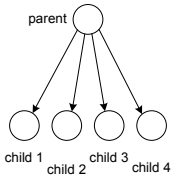


Fig. 10. The flowchart of F1 address generator



	(row, col)	row	col
parent	(62, 24)	00111110	00011000
child 1	(124, 48)	01111100	00110000
child 2	(124, 49)	01111100	00110001
child 3	(125, 48)	01111101	00110000
child 4	(125, 49)	01111101	00110001

Fig. 11. Ancestor-descendant relations of node coordinates

(C) F3: Descendant address generation

In *spatial orientation tree encoding* step, TSIHT uses DFS method to traverse all the nodes of the spatial orientation tree. It first searches the root node and each one of its branching to its immediate descendants until to the leaves. As similar to F2, the descendant address may be obtained by left-shifting one bit with adding certain necessary values.

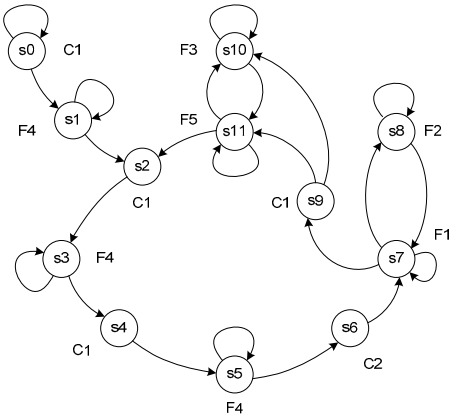
(D) F4: General linear counter

Within the first three steps of the TSIHT coding algorithm, AG behaves a general two-dimensional counter. When AG works in mode F4, the address of scanning line is generated row-by-row and column-by-column sequentially.

(E) F5: Neighbor address generation

The addresses of the four neighbor nodes originated from the same ancestor have the same property that their row or column addresses are identical except the last bit. And, their address pairs (*row,col*) of the last bit are variety with following sequence (0,0) $\rightarrow$ (0,1) $\rightarrow$ (1,0) $\rightarrow$ (1,1). When AG works in mode F5, the neighbor addresses of each node can be generated by using such principle. Since, each iteration of TSIHT coding algorithm ends at F5, after PIE finishes working at mode F5, an iteration flag signal *It_flag* is produced to notify other control units.

When PIE performs TSIHT coding algorithm, AG generates the addresses of the coefficients with coordinate pair (*row,col*) using one of above function units to access the coefficient or tag memory. Only one function unit is allowed to read input data and execute its task each time. At the front of each function unit, a latch is added to reduce the power consumption as showing in Figure 8. Besides, before entering one function from others, it may also need to clear previous state. All these function units are controlled by *AG_controller*. Let *C1* and *C2* be the clear states, and {s0, s1,…, s11} be the control state set of AG controller. The finite state machine of *AG_controller* is illustrated in Figure 12. It also shows the states, functions and the corresponded proceeding stages of TSIHT coding.



state	func.	proceeding stage
s0	idle	
s1	F4	Step 1. Initialization
s2	C1	
s3	F4	Step 2. Refinement output
s4	C1	
s5	F4	Step 3. TIP testing
s6	C2	
s7	F1	Step 4. TST update
s8	F2	
s9	C1	
s10	F3	Step 5. Spatial orientation tree encoding
s11	F5	

Fig. 12. Finite state machine of *AG_controller*

(2) *Threshold Generator* (TG): TG is used to generate initial threshold, $n = \lfloor \log_2(\max\{|c_{i,j}|\}) \rfloor$, from all coefficients and to generate the value *n* at every iteration in TSIHT coding. The hardware architecture of TG is illustrated in Figure 13. TG first reads all the coefficients and performs *or* operation bit-by-bit to find the maximum coefficient and store it in the buffer. After finding the maximum coefficient, Leading Zero Detector is used to find the position of most significant bit (MSB) to obtain the initial value *n*. Then, count-down counter continually decreases *n* by 1 and outputs the value to other circuits at every iteration.

(3) *Bit-stream Generator* (BG): In PIE, Bit-stream Generator (BG), as showing in Figure 14, generates the encoded bit stream bit-by-bit. The primary component, *Significance Test Unit*, of BG is used to check whether a coefficient is significant is significant or not. According to the TSIHT algorithm, BG outputs values depend on threshold, TST signal, magnitude and sign of coefficient. The output signals of BG include the *bit_out* bit stream and synchronous *sync* signals. Note that, only when *sync* asserts, the bit stream appearing at *bit_out* signal is meaningful.

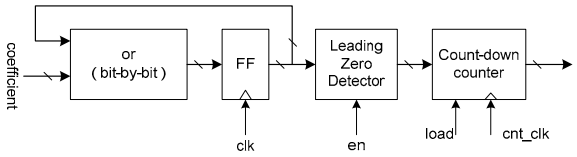


Fig. 13. Threshold Generator

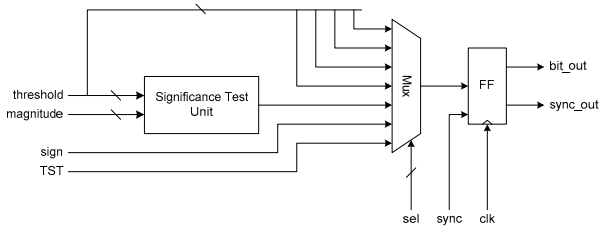


Fig. 14. Bit-stream Generator

(4) *Tag Access Unit* (TAU): To store three two-dimensional tag arrays, two 256×256 bits and one 128×128 bits RAM blocks are needed and controlled by *Tag Access Unit*. In this work, each tag memory is 8 bits wide; however, each tag flag is a one-bit data. To access each bit from 8 bits wide memory using 16-bit address signal, **Addr[15:0]**, TAU uses a similar architecture shown in Figure 15. When TAU reads one bit from tag memory, it first generates a 13-bits address signal, **Addr[15:3]**, to read one byte data, then it uses the lowest 3-bits address signal, **Addr[2:0]**, to indicate that one-bit tag. When TAU writes one bit of tag memory, it first reads the mentioned bytes as reading operation, then it replaces that one-bit tag to tag memory. Thus, TAU needs one clock cycle to read each bit and two clock cycles to write it.

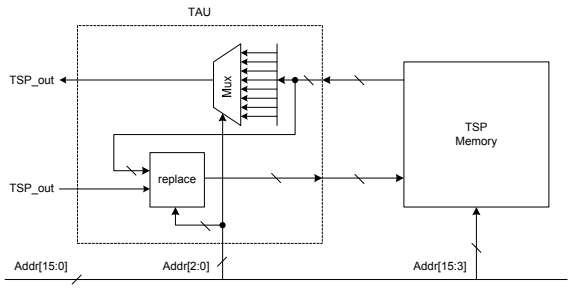


Fig. 15. TSP memory access in Tag Access Unit

(5) *Clock Divider* (CD): As mentioned in previous sections, TAU needs one clock cycle for reading operation and two clock cycles for writing. Besides, AG also needs at most three clock cycles to output encoded bit stream including value '1', sign of coefficient and TST value when it finds a coefficient is significant. Thus, PIE needs three different clock frequencies in hardware circuit. In this work, *Clock Divider* generates three clocks using divide-by-2 and divide-by-8 circuits. Figure 16 illustrates these clocks, **clk1**, **clk2** and **clk3** respectively.

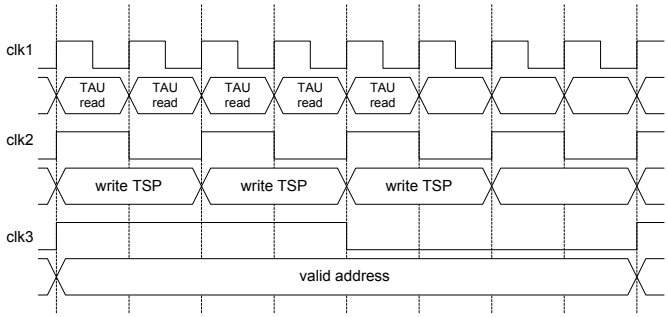


Fig. 16. Three different working clocks

VI. VLSI Architecture of TSIHT Encoder

A. Synthesis Result

A prototype of a 256×256 gray-scale image PIE core for progressive image transmission has been designed using standard cells in a semi-custom methodology. The PIE core has been synthesized with VHDL based top-down design flow and implemented by using a 0.35-μm one-poly-four-metal CMOS technology. The chip has an area of 550 μm × 450 μm = 247500 μm^2, where the AG accounts for 66% of the total surface. Figure 17 illustrates the IC layout of the PIE core. The gate count statistics of each circuit component is illustrated in Table I. The hardware characteristics of PIE core are shown in Table II.

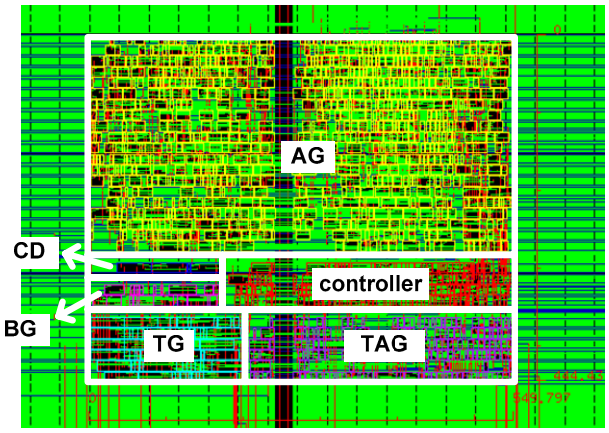


Fig. 17. PIE core layout

TABLE I
GATE COUNT STATISTICS

	AG	TG	TAU	FSM	BG	CD	Total
#.	1687	264	264	255	56	34	2560
%	66%	10%	10%	11%	2%	1%	

B. Performance

Table III lists the latency of the critical path and the gate count of the proposed PIE core and other encoders. The gate count of EZT encoder in [13] is reported about 5 *K* gates, it is almost twice as proposed PIE core. The gate count of EZW encoder in [14] in about 3889 gates, and the latency of the critical path is 16.53 *ns*. Although, the handling image size of proposed PIE is less than others. While considering larger image size implementation, PIE core only increases the memory size but few the gate count of the circuit. Moreover, the latency of the PIE core is less than EZT in [13] and EZW encoder in [14]. Thus, PIE core is a faster and simpler architecture than others.

TABLE 2
HARDWARE CHARACTERISTIC OF PIE CORE

Property	Result
Technology	0.38 μm 1P4M CMOS
Area	247500 μm^2
Total gate count	2560
Latency	6.32 *ns*
Max. working freq.	158 *MHz*
Image size	256×256
Pin count	90

TABLE III
PERFORMANCE COMPARISON

	EZT [13]	EZW [14]	PIE
Image size	352×288	720×480	256×256
Latency	N/A	16.53 *ns*	6.32 *ns*
Gate count	5000	3889	2560

While considering memory usage, as illustrating in Table IV, both SPIHT [4] and intra-band partitioning [15] need more than one hundred *K* bytes memory. The proposed TSIHT coding only occupies about 18 *K* bytes, which is less than other coding algorithms.

TABLE IV
MEMORY USAGE

	SPIHT [4]	Karam's [15]	TSIHT
Image size	256×256	256×256	256×256
Memory	250 *K* bytes	312.5 *K* bytes	18 *K* bytes

VII. Conclusion

Although, SPIHT [4] coding is approved that it is the most efficient algorithm to implement EZW coding, the problem of large amount memory occupied may restrict its applications. In this work, proposed TSIHT coding using tag flags can effectively reduce amount of memory usage. For a typical 256×256 gray-scale image, TSIHT only needs 18 *K* bytes, while SPIHT needs 250 *K* bytes memory, which is almost 13.9 times the proposed TSIHT coding. The VLSI implementation, PIE core, also provides a lower gate count (about 2560), smaller area (247500 μm^2) and higher speed (158 *MHz* at Max.) circuit, which is convenient to be integrated into other image compression systems.

Acknowledgment

This work was supported in part by the National Science Council, Taiwan, under the grant number NSC 95-2221-E009-337-MY3 and Chung-Shan Institute of Science and Technology, Taiwan, under the project BV94G10P. The authors would like to thank National Chip Implementation Center (CIC) for technical support.

REFERENCES

[1] ISO/IEC, *JPEG 2000 Committee Draft version 1.0, cd15444-1 edition*, Dec. 1999.

[2] C. Christopoulos, A. Skodras, and T. Ebrahimi, "The JPEG2000 still image coding system: An overview," *IEEE Transactions on Consumer Electronics*, vol. 46, pp. 1103–1127, Nov. 2000.

[3] T. Sikora, "The MPEG-4 video standard verification model," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 1, pp. 19–31, Feb. 1997.

[4] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing*, vol. 41, pp. 3445–3462, Dec. 1993.

[5] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243–250, June 1996.

[6] D. Mukherjee and S. K. Mitra, "Vector spiht for embedded wavelet video and image coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 3, pp. 231–246, Mar. 2003.

[7] Z. Wang and A. C. Bovik, "Embedded foveation image coding," *IEEE Transactions on Image Processing*, vol. 10, no. 10, pp. 1397–1410, Oct. 2001.

[8] T. Kim, S. Choi, R. E. V. Dyck, and N. K. Bose, "Classified zerotree wavelet image coding and adaptive packetization for low-bit-rate transport," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 9, pp. 1022–1034, Sept. 2001.

[9] W. A. Pearlman, A. Islam, N. Nagaraj, and A. Said, "Efficient, low-complexity image coding with a set-partitioning embedded block coder," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 11, pp. 1219–1228, Nov. 2004.

[10] A. Munteanu, J. Cornelis, G. V. der Auwera, and P. Cristea, "Wavelet image compression - the quadtree coding approach," *IEEE Transactions on Information Technology in Biomedicine*, vol. 3, no. 3, pp. 176–185, Sept. 1999.

[11] S. G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674–693, July 1989.

[12] S.-F. Hsiao, Y.-C. Tai, and K.-H. Chang, "Vlsi design of an efficient embedded zerotree wavelet coder with function of digital watermarking," *IEEE Transactions on Consumer Electronics*, vol. 46, no. 7, pp. 628–636, Aug. 2000.

[13] B. Vanhoof, M. Peon, G. Lafruit, J. Bormans, M. Engels, and I. Bolsens, "A scalable architecture for mpeg-4 embedded zero tree coding," *Custom Integrated Circuit Conference*, pp. 65–68, 1999.

[14] R. Y. OMAKI, G. FUJITA, T. ONOYE, and I. SHIRAKAWA, "Architecture of embedded zerotree wavelet based real-time video coder," *Proceedings 12th IEEE ASIC/SOC Conference*, pp. 137–141, 1999.

[15] Z. Liu and L. J. Karam, "An efficient embedded zerotree wavelet image codec based onintraband partitioning," *IEEE International Conference on Image Processing*, vol. 3, pp. 162–165, Sept. 2000.

Tsung-Hsi Chiang (SM'00) received the B.S. degree in Electrical Engineering, I-Shou University, Kauhsiung, Taiwan in 1998 and the master degree in Mechanical Engineering, Yuan-Ze University, Taoyuan, Taiwan in 2000. His research interests are VLSI architecture, coding theory and image processing.

Lan-Rong Dung (SM'93-M'97) received a BSEE and the Best Student Award from Feng Chia University, Taiwan, in 1988, an MS in electronics engineering from National Chiao Tung University, Taiwan, in 1990, and Ph.D. in electrical and computer engineering from Georgia Institute of Technology, in 1997.

From 1997 to 1999 he was with Rockwell Science Center, Thousand Oaks, CA, as a Member of the Technical Staff. He joined the faculty of National Chiao Tung University, Taiwan in 1999 where he is currently an associate professor in the Department of Electrical and Control Engineering. He received the VHDL International Outstanding Dissertation Award celebrating in Washington DC in October, 1997. His current research interests include VLSI design, digital signal processing, hardware-software codesign, and System-on-Chip architecture. He is a member of Circuits and Systems and Signal Processing societies of the IEEE.