PAPER *Special Section on VLSI Design and CAD Algorithms*

VLSI Implementation of a Modified Efficient SPIHT Encoder

Win-Bin HUANG[†a), *Student Member*, **Alvin W. Y. SU**[†], *and* **Yau-Hwang KUO**[†], *Nonmembers*

SUMMARY Set Partitioning in Hierarchical Trees (SPIHT) is a highly efficient technique for compressing Discrete Wavelet Transform (DWT) decomposed images. Though its compression efficiency is a little less famous than Embedded Block Coding with Optimized Truncation (EBCOT) adopted by JPEG2000, SPIHT has a straight forward coding procedure and requires no tables. These make SPIHT a more appropriate algorithm for lower cost hardware implementation. In this paper, a modified SPIHT algorithm is presented. The modifications include a simplification of coefficient scanning process, a 1-D addressing method instead of the original 2-D arrangement of wavelet coefficients, and a fixed memory allocation for the data lists instead of a dynamic allocation approach required in the original SPIHT. Although the distortion is slightly increased, it facilitates an extremely fast throughput and easier hardware implementation. The VLSI implementation demonstrates that the proposed design can encode a CIF (352 × 288) 4:2:0 image sequence with at least 30 frames per second at 100-MHz working frequency.

key words: *discrete wavelet transform (DWT), set partitioning in hierarchical trees (SPIHT), image coding*

1. Introduction

Discrete Wavelet Transform (DWT) has been widely used for digital image compression [1], [2]. Bi-orthogonal (5,3) and (9,7) filters were chosen to be the standard filters used in the JPEG 2000 codec standard [3], [4]. Since DWT was introduced, several codec algorithms were proposed to compress the transform coefficients as much as possible. Among them, Embedded Zerotree Wavelet (EZW), Set Partitioning In Hierarchical Trees (SPIHT) and Embedded Block Coding with Optimized Truncation (EBCOT) are the most famous ones [5], [6]. In [6], if no entropy coding or arithmetic coding methods are incorporated, coding tables are not required with slight loss in compression ratio. Moreover, SPIHT can be easily used for either fixed bit rate or variable bit rate applications, and it is also very suitable for progressive transmission [5]. Furthermore, SPIHT has about 0.6 dB peak-signal-to-noise-ratio (PSNR) gain over EZW [7] and is very close to EBCOT in many circumstances [6].

While EBCOT has the best compression rate of all, it requires complex multi-layer coding procedures, multiple coding tables and arithmetic coding techniques. So its hardware implementation would be more difficult and expensive. [10] On the contrary, SPIHT applies a much simpler coding procedure and needs no coding table. The implementation of SPIHT would be much cheaper to be suitable for still image compression appliances. Moreover, the SPIHT based encoding algorithm is also applied to the SOT based audio compression [15]–[17]. There are still some interested issues on SPIHT based algorithms and applications. For example, rate-distortion is always an important issue in data compression. Lots of considerations were focused on maximizing compression rate as well as minimizing distortion under a limited data rate. To achieve a progressive bitstream with a smooth rate-distortion curve was discussed more in the past decade for various multimedia applications over Internet. Scalability is easily achieved by SPIHT-based methods, such as image, video, or audio coders [23]–[25].

However, its implementation on a silicon chip still encounters some difficulties. First of all, the addressing scheme of the wavelet coefficients is in a 2-D manner, and the searching for the descendants of a given coefficient has to be performed very frequently. Next, frequent transactions among the three lists, *List of Insignificant Set* (LIS), *List of Insignificant Pixel* (LIP) and *List of Significant Pixel* (LSP), which store the coefficient coordinates, are all necessary. Third, linked lists are suitable for the implementation of the three lists, and insertion or deletion operations are necessary to update those lists. Yet linked lists require more memory space [14] and are more time-consuming in search, insertion and deletion operation. Nevertheless, for VLSI realization, it is necessary to develop fast hardwired circuit solutions to speed up the time-consuming computations mentioned above.

Based on the above observations, a modified SPIHT suitable for hardware implementation is proposed. In spite of special arrangement, the coefficients can be stored in a 1-D memory array. The searching for the descendants of each coefficient becomes very simple. Besides, instead of using dynamic linked lists, fixed size tables are used to execute the operations for the three lists. Transactions of the lists require no storage of the coordinates, and only the special-purposed flags stored in the tables are updated. Inspection of the content of the three lists becomes very convenient. In summary, by the modified SPIHT, the overall coding procedure is simplified and efficient. The only disadvantage is that the average PSNR is decreased by about 0.2 dB occasionally. The VLSI implementation of the modified SPIHT can encode a CIF (352×288) 4:2:0 image sequence with at least 30 frames per second at 100-MHz working frequency. The

Manuscript received March 17, 2006.
Manuscript revised June 12, 2006.
Final manuscript received August 1, 2006.

[†]The authors are with the Department of Computer Science and Information Engineering, National Cheng-Kung University, Tainan, Taiwan, R.O.C.
a) E-mail: huangwb@cad.csie.ncku.edu.tw

DOI: 10.1093/ietfec/e89–a.12.3613

overall gate count is merely 2950 and the internal memory is 4 kb for storing the three lists and wavelet coefficients using TSMC 0.35-μm technology. Since SPIHT can not only be applied to image compression but also audio and video compression [23]–[25], it is possible to handle both video and audio bit-streams in one single hardware module with the proposed architecture in this paper.

This paper is organized as follows. The modified SPIHT is presented in Sect. 2. VLSI implementation of the modified SPIHT is discussed in Sect. 3. Finally, the conclusion is shown in Sect. 4.

2. Modified SPIHT

2.1 Motivation

The original SPIHT is presented in [5] by A. Said and W.A. Pearlman. Searching for descendants of a specific coefficient takes lots of operations. For a dedicated hardware, it is necessary to store 2-D data in a 1-D array. Without loss of generality, a 4 resolution level 16-by-16 DWT decomposed image is used as an example. There are 256 coefficients to be encoded. Figure 1 illustrates a typical addressing method.

For example, direct off-springs of a coefficient at the position-002 are at position-004, position-005, position-020, and position-021 while other descendants are at position-008 to position-011, position-024 to position-027, position-040 to position-043 and position-056 to position-059. The rule is as follows.

Address of direct off-spring No.0

$$= (\textit{Address of the parent node} \times 2) \quad (1)$$

Address of direct off-spring No.1

$$= (\textit{Address of the parent node} \times 2) + 1 \quad (2)$$

Address of direct off-spring No.2

$$= (\textit{Address of the parent node} \times 2) + 16 \quad (3)$$

Address of direct off-spring No.3

$$= (\textit{Address of the parent node} \times 2) + 17 \quad (4)$$

The above operations are not complicated at all. However, it is not efficient to implement the dedicated SPIHT hardware by using the rule. One can simplify the design by altering the addressing method. In 16-by-16 cases, 256 addresses are needed. One can use 8-bit symbols to represent 2-D addresses. In Fig. 2 (left), the upper nibble is for y-axis (height) and the lower nibble is for x-axis (width).

A new addressing is used as shown in Fig. 2 (right). For instance, the coefficient stored in position-071 (b'0100 0111) is stored in position-053 (b'00110101). This notation yields the physical address in Fig. 3. Because searching for the descendants of a given coefficient has to be performed very frequently, it is necessary to design a dedicate circuit to compute the addresses. This also consumes more clock cycles. In Fig. 3, the numbers in brackets represent the original addresses of the wavelet coefficients, which are also shown in Fig. 1. Moreover, the numbers not in brackets indicate the new storage addresses of the wavelet coefficients. The thick lines are used to divide the sub-bands like the thick lines in Fig. 1. For example, the new position of the coefficient at position-002 in Fig. 1 is at position-004 in Fig. 3, and its direct off-springs are at position-016, position-017, position-018, and position-019 while other descendants are at position-064 to position-067, position-068 to position-071, position-072 to position-075 and position-076 to position-079.

Therefore, with the proposed addressing method, searching for the descendants of a specific coefficient becomes easier. The rule in Eqs. (1)–(4) is modified as follows.

New Address of direct off-spring No.0

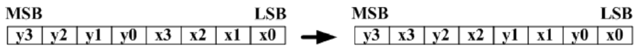

Fig. 2 Original addressing method (left) and modified addressing method (right).

000	001	002	003	004	005	006	007	008	009	010	011	012	013	014	015
016	017	018	019	020	021	022	023	024	025	026	027	028	029	030	031
032	033	034	035	036	037	038	039	040	041	042	043	044	045	046	047
048	049	050	051	052	053	054	055	056	057	058	059	060	061	062	063
064	065	066	067	068	069	070	071	072	073	074	075	076	077	078	079
080	081	082	083	084	085	086	087	088	089	090	091	092	093	094	095
096	097	098	099	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Fig. 1 16-by-16 DWT coefficients with normal address method.

000 (000)	001 (001)	002 (016)	003 (017)	004 (002)	005 (003)	006 (018)	007 (019)	008 (032)	009 (033)	010 (048)	011 (049)	012 (034)	013 (035)	014 (050)	015 (051)
016 (004)	017 (005)	018 (020)	019 (021)	020 (006)	021 (007)	022 (022)	023 (023)	024 (036)	025 (037)	026 (052)	027 (053)	028 (038)	029 (039)	030 (054)	031 (055)
032 (064)	033 (065)	034 (080)	035 (081)	036 (066)	037 (067)	038 (082)	039 (083)	040 (096)	041 (097)	042 (112)	043 (113)	044 (098)	045 (099)	046 (114)	047 (115)
048 (068)	049 (069)	050 (084)	051 (085)	052 (070)	053 (071)	054 (086)	055 (087)	056 (100)	057 (101)	058 (116)	059 (117)	060 (102)	061 (103)	062 (118)	063 (119)
064 (008)	065 (009)	066 (024)	067 (025)	068 (010)	069 (011)	070 (026)	071 (027)	072 (040)	073 (041)	074 (056)	075 (057)	076 (042)	077 (043)	078 (058)	079 (059)
080 (012)	081 (013)	082 (028)	083 (029)	084 (014)	085 (015)	086 (030)	087 (031)	088 (044)	089 (045)	090 (060)	091 (061)	092 (046)	093 (047)	094 (062)	095 (063)
096 (072)	097 (073)	098 (088)	099 (089)	100 (074)	101 (075)	102 (090)	103 (091)	104 (104)	105 (105)	106 (120)	107 (121)	108 (106)	109 (107)	110 (122)	111 (123)
112 (076)	113 (077)	114 (092)	115 (093)	116 (078)	117 (079)	118 (094)	119 (095)	120 (108)	121 (109)	122 (124)	123 (125)	124 (110)	125 (111)	126 (126)	127 (127)
128 (128)	129 (129)	130 (144)	131 (145)	132 (130)	133 (131)	134 (146)	135 (147)	136 (160)	137 (161)	138 (176)	139 (177)	140 (162)	141 (163)	142 (178)	143 (179)
144 (132)	145 (133)	146 (148)	147 (149)	148 (134)	149 (135)	150 (150)	151 (151)	152 (164)	153 (165)	154 (180)	155 (181)	156 (166)	157 (167)	158 (182)	159 (183)
160 (192)	161 (193)	162 (208)	163 (209)	164 (194)	165 (195)	166 (210)	167 (211)	168 (224)	169 (225)	170 (240)	171 (241)	172 (226)	173 (227)	174 (242)	175 (243)
176 (196)	177 (197)	178 (212)	179 (213)	180 (198)	181 (199)	182 (214)	183 (215)	184 (228)	185 (229)	186 (244)	187 (245)	188 (230)	189 (231)	190 (246)	191 (247)
192 (136)	193 (137)	194 (152)	195 (153)	196 (138)	197 (139)	198 (154)	199 (155)	200 (168)	201 (169)	202 (184)	203 (185)	204 (170)	205 (171)	206 (186)	207 (187)
208 (140)	209 (141)	210 (156)	211 (157)	212 (142)	213 (143)	214 (158)	215 (159)	216 (172)	217 (173)	218 (188)	219 (189)	220 (174)	221 (175)	222 (190)	223 (191)
224 (200)	225 (201)	226 (216)	227 (217)	228 (202)	229 (203)	230 (218)	231 (219)	232 (232)	233 (233)	234 (248)	235 (249)	236 (234)	237 (235)	238 (250)	239 (251)
240 (204)	241 (205)	242 (220)	243 (221)	244 (206)	245 (207)	246 (222)	247 (223)	248 (236)	249 (237)	250 (252)	251 (253)	252 (238)	253 (239)	254 (254)	255 (255)

Fig. 3 16-by-16 DWT coefficients with new addressing method.

$$= (\textit{New Address of the parent node} \times 4) \quad (5)$$

$$\textit{New Address of direct off-spring No.1}$$
$$= (\textit{New Address of the parent node} \times 4) + 1 \quad (6)$$

$$\textit{New Address of direct off-spring No.2}$$
$$= (\textit{New Address of the parent node} \times 4) + 2 \quad (7)$$

$$\textit{New Address of direct off-spring No.3}$$
$$= (\textit{New Address of the parent node} \times 4) + 3 \quad (8)$$

Thus, the direct off-springs of a specific coefficient are stored in consecutive addresses. This makes the memory-read operation more efficient and could reduce the switching frequency of the address bus as well. The address generation circuit is reduced to a 2-bit shifter and an increment-by-one operation. Hence, a new coding algorithm, which can simplify the hardware design without losing much coding efficiency, is derived and explained in next section. Most image compression standards, e.g., JPEG2000, suggest that images should be divided into code blocks, and the size of code blocks is limited to powers of two with the minimum size being 2^2 and the maximum being 2^{10} [22]. Besides the width and height of code block are usually the same for designing a dedicated and efficient hardware. Under the conditions the rule in Eqs. (5)–(8) is always held, and this is proved in the Appendix A.

2.2 Modified SPIHT Algorithm

Most definitions of the lists and symbols are identical to the original SPIHT except that 1-D addresses are used instead of 2-D addresses and three definitions, that is, $MaxLIP$, $MaxLIS$ and $MaxLSP$, are increased. They are as follows:

- $O(i)$: Set of coordinates of all offspring of node i.
- $D(i)$: Set of coordinates of all descendants of node i.
- H: Set of coordinates of all spatial orientation tree roots.
- $L(i)$:$D(i) - O(i)$.
- c_i: The magnitude of the coefficient of node i.
- $MaxLIP$: The maximal address of all nodes in the LIP list.
- $MaxLIS$: The maximal address of all nodes in the LIS list.
- $MaxLSP$: The maximal address of all nodes in the LSP list.
- T: All nodes in the image.
- N: The address of the last node in the image.
- A set $L(i)$ or $D(i)$ is said to be significant if any coefficient in the set has a magnitude greater than the threshold, such as

$$S_n(G) = \begin{cases} 1, & max\{|\ c_i\ |\}, \forall(i) \in G \\ 0, & \text{otherwise.} \end{cases} \quad (9)$$

where G, $G \subseteq T$, is a set of nodes.

Similar to the SPIHT algorithm, four encoding steps, initialization, sorting pass, refinement pass and quantization pass, are performed and three linked lists, LIS, LSP and LIP, are used in the proposed SPIHT. Its pseudo-code is described as follows.

```
1. Initialization step:
  • Output $n = \lfloor \log_2(max\{|\ c_i\ |\}) \rfloor, 0 \le i \le N$;
  • Set $LSP(i) = 0, 0 \le i \le N$;
  • Set $LIP(i) = 1, \forall i \in H$, otherwise set to 0;
  • Set $LIS(i) = A, \forall i \in H$ with descendants,
    otherwise set to 0;
  • Set $MaxLIP = max(H), MaxLIS = max(H)$,
    $MaxLSP = 0$;
2. Refinement pass:
  • For $0 \le i \le MaxLSP$
      - If $LSP(i) = 1$ then
          Output the $n$th most significant bit of $|\ c_i\ |$;
      - Else if $LSP(i) = 2$ then
          Set $LSP(i) = 1$;
3. Sorting pass:
  • For $0 \le i \le MaxLIP$
      - If $LIP(i) = 1$ then
          * Output $S_n((i))$,
          * If $S_n((i)) = 1$, then
              ⋄ Output sign of $c_i$
              ⋄ Set $LSP(i) = 2$,
                $MaxLSP = max(i, MaxLSP)$
              ⋄ Set $LIP(i) = 0$
  • For $0 \le i \le MaxLIS$
      - If $LIS(i) = A$ then
          * Output $S_n(D(i))$
          * If $S_n(D(i)) = 1$ then
              ⋄ For $(i \times 4) \le j \le (i \times 4 + 3)$
                  ★ Output $S_n((j))$
                  ★ If $S_n((j)) = 1$ then
                    Set $LSP(j) = 2$,
                    $MaxLSP = max(i, MaxLSP)$
                    Output the sign of $c_j$
                  ★ Else
                    Set $LIP(j) = 1$,
                    $MaxLIP = max(i, MaxLIP)$
              ⋄ If $(i \times 16) < N$ then
                Set $LIS(i) = B$,
                $MaxLIS = max(i, MaxLIS)$
              ⋄ Else
                Set $LIS(i) = 0$
      - If $LIS(i) = B$ then
          * Output $S_n(L(i))$
          * If $S_n(L(i)) = 1$, then
              ⋄ For $(i \times 4) \le j \le (i \times 4 + 3)$
                Set $LIS(j) = A$
              ⋄ Set $LIS(i) = 0$
4. Quantization-step update pass:
  • Decrement n by one and go to Step 2.
```

In both SPIHT algorithms, there are three lists, LIS, LIP, and LSP to be constructed. When an element in LIS changes its type, transactions among the lists are necessary. Insertion and deletion operations of the lists are required, too. Dynamic linked lists have to be employed to construct the three lists for the SPIHT algorithm. It is easy to construct dynamic linked lists with high level programming languages because memory management mechanism is available in computer operating systems. However, design of such a dedicated low cost circuit is not so straightforward. These operations reduce the throughput at the same time. With the proposed approach, no dynamic linked list is necessary. If N is the total number of coefficients, based on the proposed addressing method, addresses of the coefficients which have descendants are from 0 to $(N/4)-1$. Tables with $N/4$ entries are used for the lists. They are $LIS(i)$, $LIP(i)$, and $LIS(i)$, for $i = 0, \ldots, (N/4)-1$. $LIS(i)$ is set to be zero when node-i has no descendant, or it has not joined the encoding process yet. Otherwise, it denotes a coefficient at position-i as either type A or type B. Similar definitions are used for $LIP(i)$ and can be found in the algorithm. $LSP(i)$ is set to be zero when node-i is insignificant pixel. $LSP(i)$ is set to be 2 when node-i becomes significant pixel for the first time, then it is set to be 1 after the node-i passes the refinement pass. By scanning the tables subsequently, one encoding pass is finished. Simple counters and finite state machines (FSM) are enough for its implementation. The test images are 8 bpp grey scale images. Bi-orthogonal (9,7) filter and 5-level DWT are used [2]. The other pre-processing follows the suggestions in JPEG2000 [4]. The distortion measure is calculated by Eq. (10).

$$PSNR = 10\log_{10}\left(\frac{255^2}{MSE}\right) \tag{10}$$

where MSE denotes the mean-square-error between the original image and the decoded image. No entropy coding is employed.

Figure 4 presents the rate-distortion comparison among the three algorithms, the original SPIHT [5], the modified SPIHT [11] and the proposed SPIHT. The solid blue lines exhibit rate vs. distortion performance for SPIHT without arithmetic coding. The solid red lines show the performance of the proposed SPIHT without arithmetic coding. The dash-dot blue lines demonstrate the performance of the modified SPIHT, which is an old version of the proposed SPIHT.

All decoded images were recovered from a single fidelity embedded encoded file, truncated at the desired rate. Obviously, the PSNR performance of the proposed SPIHT is much higher than the modified SPIHT at all of the rate conditions. Notice that the performances of the original SPIHT and the proposed SPIHT. At rates below 1.0 bpp, the difference falls within 0.2 dB, and it becomes negligible at lower rates. The difference increases and falls within 0.3 dB between the rate 1.0 bpp and 1.8 bpp. Since most applications demand higher compression ratios (more than 16:1), the proposed algorithm should be acceptable. Because linked lists are used in the original SPIHT, the nodes that turn significant earlier will be placed near the heads of the lists. These nodes are more important than the nodes that turn significant later. Therefore, the bits for the earlier nodes will also be output first. The new algorithm has no such scheme. That is, the original SPIHT tends to have better PSNR performance than the proposed algorithm. Fortunately, PSNR performance of the new algorithm does not decrease much. No arithmetic coding was used on the significant test for these results. Back-end arithmetic coding using contexts and joint encoding generally improves SPIHT by about 0.5 dB [14]. We may expect that improvement for the proposed SPIHT as well.

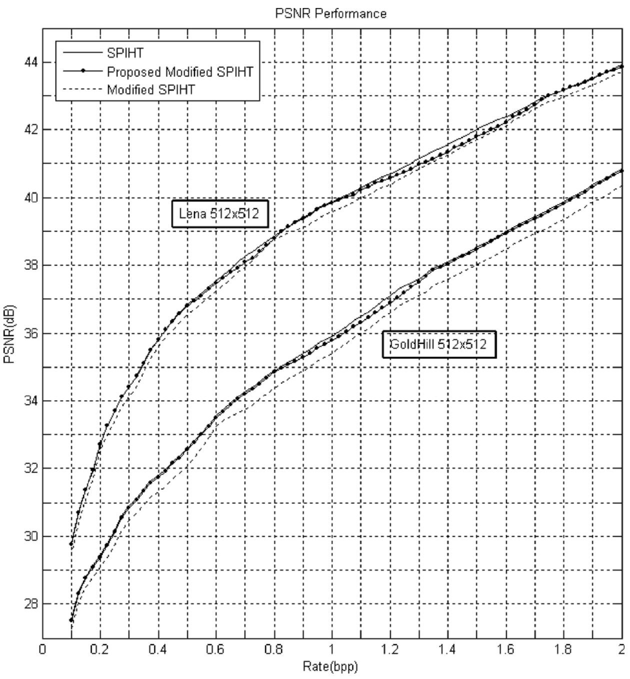

Fig. 4 Coding performances for original SPIHT [5] (line), the proposed modified SPIHT (line with dots) and the modified SPIHT [11] (dotted line) on LENA and GOLDHILL images.

Table 1 CPU times in ms of these SPIHT algorithms.

Rate	Original SPIHT [5]	Modified SPIHT [11]	Proposed SPIHT
0.2 bpp	647	98	100
0.4 bpp	673	105	107
0.6 bpp	699	107	111
0.8 bpp	724	115	118
1.0 bpp	731	118	121

Though the target of the proposed algorithm is designed for low cost ASIC implementation, the improvement in speed on a general purpose computer can be expected. Excluding the time spent for I/O and 8-level DWT, the corresponding CPU times of two algorithms for encoding LENA at different rates are shown in Table 1. The result is obtained by taking the average after executing the programs 1000 times. The programs run on a Pentium IV-2G with 512 MB memory. The new method is over 6 times faster than SPIHT.

3. The Proposed Architecture

In this paper, 16 × 16 image blocks are used as a design ex-

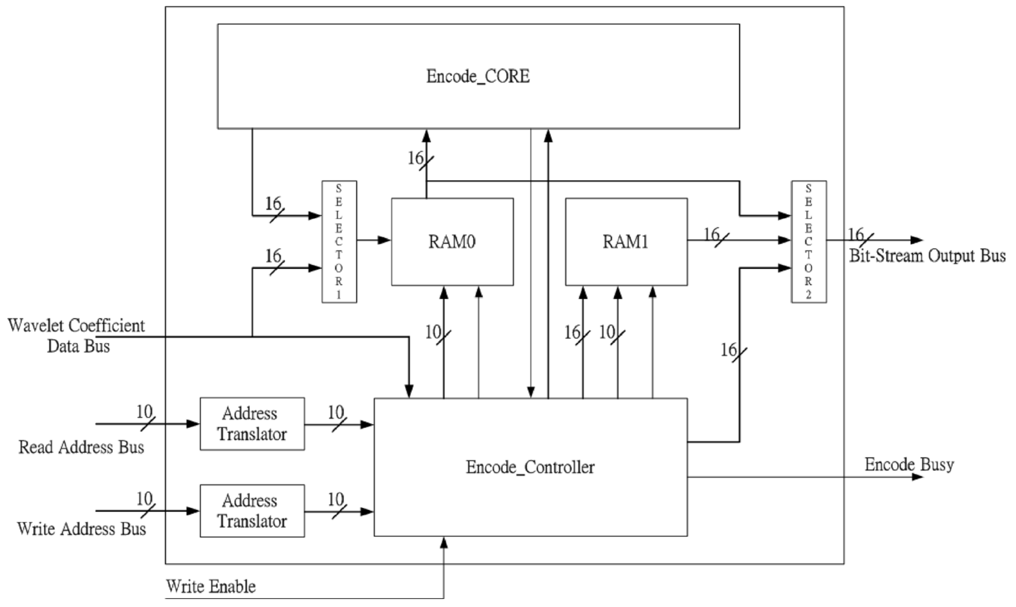

Fig. 5 The proposed SPIHT architecture.

Table 2 Address space used in encode hardware.

Address	Usage
x000–x0ff	The Wavelet Coefficient
x100–x1ff	The Output Bit-Stream
x200	Bit-Stream Length
x201	Encode Stop Level and Encode Point

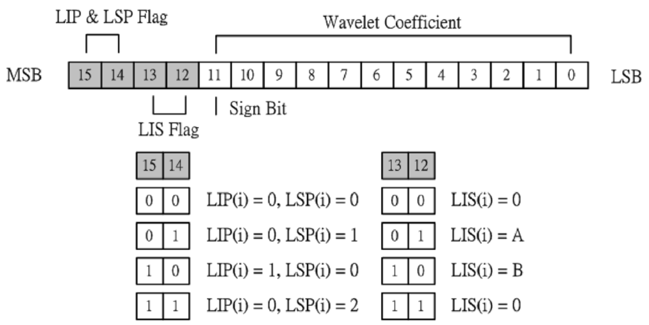

Fig. 6 Data format of the content of each node.

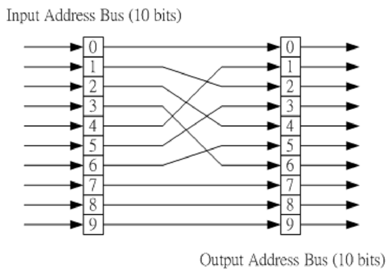

Fig. 7 Address translator.

ample. Implementations for different sizes can be extended easily. The architecture of the encoding engine, includes a core module, a system controller, two bus selectors, two address translators and two internal memory buffers. This is shown in Fig. 5.

3.1 The Proposed Encoding Engine

The proposed encoding engine is treated as the peripheral of a certain CPU. The memory-mapping I/O is adopted in this architecture for handling I/O. The engine and the memory share the memory map displayed in Table 2.

The first 256 address spaces are for the wavelet coefficients. The output bit-stream is stored in the second local memory with the next 256 address spaces. The information of bitstream length, Stop_Level and Stop_Point is recorded. The Stop_Level is in the bit plane where the encoding process stops. Similarly, the Stop_Point is the coordinate at which the encoding process stops.

After performing Bi-Orthogonal (9, 7) or (5, 3) DWT decomposition, the coefficients are carried into the local memory of the encoding engine. What is more, the coefficient of each node should be stored using the format shown in Fig. 6. The first four bits are used to record the transactions of the lists that happened in the coding passes. The remaining bits indicate the results after each coding pass. Based on the proposed algorithm, the coefficients should be stored as in Fig. 3. The address is translated using the translator shown in Fig. 7.

The function of the Encode_Controller module includes handling of the message from the CPU. Two states are performed in Encode_Controller module: The first state is to wait for the completion of transfer of wavelet coefficients or bit-streamand the second state indicates that the engine is in the encoding process.

The source of the data for the first local memory is selected using the circuit in Fig. 8. Likewise, there are three sources for data output bus and the selection is done by us-

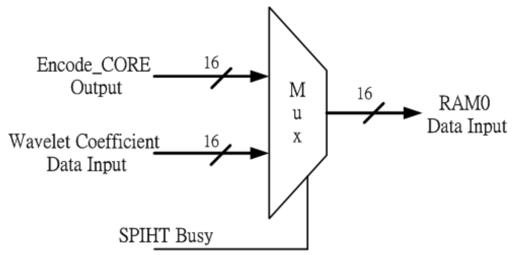

Fig. 8 Data selector 1.

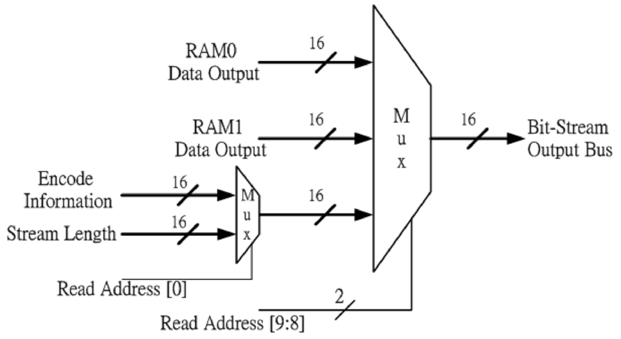

Fig. 9 Data selector 2.

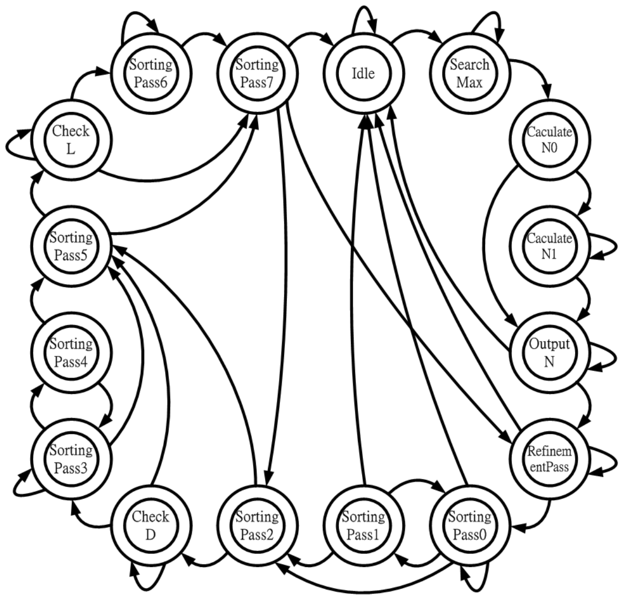

Fig. 10 State diagram of the Encode_CORE module.

ing the circuit in Fig. 9.

The Encode_CORE module consisted of a finial state machine is the major processing unit in Fig. 5. It is then used to control the dataflow of the modified SPIHT algorithm and generate the final bit-stream. Therefore, the final state machine of the Encode_CORE module implements the algorithm in Sect. 2.2 and its state diagram is shown in Fig. 10. The operation of each state is explained in the Appendix B. It outputs the bit-stream bit by bit to the Encode_Controller module. The bit-stream is carried into the

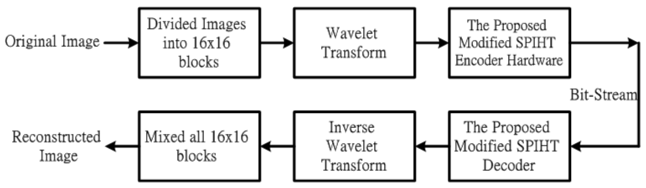
Original Image
Divided Images into 16x16 blocks
Wavelet Transform
The Proposed Modified SPIHT Encoder Hardware
Bit-Stream
Reconstructed Image
Mixed all 16x16 blocks
Inverse Wavelet Transform
The Proposed Modified SPIHT Decoder

Fig. 11 The process of the wavelet-based compression system.

Table 3 Number of clock cycles used for 16 × 16 block.

Stop_Level	Average Clock Cycles
0	5530
1	4735
2	3962
3	3255
4	2573

Table 4 Average PSNR performance of test images.

Stop_Level	PSNR
0	49.69 dB
1	48.15 dB
2	43.05 dB
3	37.84 dB
4	33.18 dB

Table 5 PSNR performance of 'Lena' images.

Stop_Level	PSNR
0	49.695 dB
1	47.673 dB
2	42.946 dB
3	38.591 dB
4	34.290 dB

second local memory by the Encode_Controller module. After each state of the Encode_CORE, wavelet coefficients are modified and restored into the first local memory. The final bit-stream will be stored in the second local memory.

3.2 Performance of the Modified SPIHT Encoder

Ten test images, downloaded from [13], are all 256 × 256 and divided into 256 16 × 16 blocks. Bi-orthogonal (9,7) 3-level DWT are utilized. The process of the wavelet-based compression system is shown in Fig. 11.

The average required clock cycles of all 16 × 16 blocks are shown in Table 3, and the average PSNR for each Stop_Level is also shown in Table 4. For example, when the StopLevel is 0, the required clock cycle is 5530 for a 16 × 16 block, and the average PSNR is 49.69 dB. If the required quality is lower, the encoding speed would be much higher.

The PSNR performance of the 'Lena' image is shown in Table 5, and it is identical to the results derived from our previous software implementation. A rate control is employed to select the appropriate data rate for each coding block to obtain better rate-distortion performance [9]. To improve rate distortion performance, the block size can be increased and the implementation can be extended from the

Table 6 Chip specification of coding engine.

Process Technology	TSMC 0.35-μm CMOS 2P4M
Chip Size	$1.2 \times 1.2\,\text{mm}^2$
Total Gate Count	2,950 gates
Clock Frequency	100 MHz
Supply Voltage	3.3 V
Power Consumption	7.5136 mW

Table 7 Synthesis result of coding engine.

Module	Gate Count
Encode_CORE module	2173.9 gates
Encode_Controller module with others	776.1 gates

Table 8 List of memories used in the prototype chip.

Usage	Memory Size (bits)
The Wavelet Coefficients	4K
The Output Bit-Stream buffer	4K

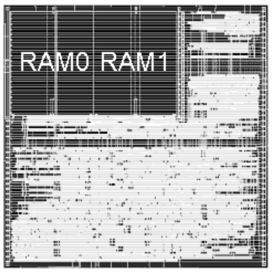

Fig. 12 Layout view of the proposed modified SPIHT engine.

current design in a very straightforward way. The design passes both pre-simulation and post-simulation tests.

Our system was desinged by using Verilog-HDL and simulated for debugging purposes with Verilog-XL. Design Analyzer from Synopsys was used to compile the Verilog-HDL code and generate a net list. The Apollo tool was used to both place and route the design. The specification of the coding engine is shown in Table 6. This design includes 6 modules, an Encode_CORE module, an Encode_Controller module, two selectors, and two address translators. In addition, two static RAMs are used in this implementation. The synthesis result of coding engine is shown in Table 7, where the address translators and selectors are synthesized with Encode_Controller. The functions of the two RAMs are presented in Table 8. The photograph is exhibited in Fig. 12. Moreover, only the address translators and RAMs need to be changed when treating different block size.

The proposed design has to be combined with DWT to become a complete image code. Bi-Orthogonal (9,7) DWT decomposition with lifting is used [4]. A reference design can be found in [11]. The design also has been completely verified on an Altera APEX PCI development kit. This board consists of an Altera APEX EP20K1000E FPGA. The Altera APEX 20K FPGA allows for 1.7 million gate designs [20]. Only 2% logic elements and 4% ram bits are required for our modified SPIHT design. It can process over 4.6 million image pixels within 1 second depending on the required image quality.

Two hardware implementations of SPIHT based algorithm are presented in [18], [19]. The SPIHT coding in [18] is performed using content addressable memories to keep track of the order in which information about the wavelet is sent for each image. No optimizations or modifications were made to the algorithm to take into account of the design that would compute on a hardware platform as opposed to a software platform. The design was simulated over an 8×8 sized image for functional verification. Since the design was only simulated, no performance numbers were given. The SPIHT coding in [19] is implemented to parallelize the computation and based upon fixed-order SPIHT, which is developed specifically for the use within adaptive hardware. For an $N\times N$ image fixed-order SPIHT, it may be calculated in $N^2/4$ cycles. However, their SPIHT design required 98% FPGA area to be implemented in Xilinx Virtex 2000E FPGA. The Xilinx Virtex 2000E FPGA allows for two million gate designs [21]. It is apparent that our system requires less FPGA area and may be a good option for low cost applications.

4. Conclusion

An efficient VLSI implementation of the modified SPIHT encoder is presented. New coefficient addressing method, the fixed-size list tables and straightforward coding procedure are employed. Low-cost and simple hardware implementation is achieved. Though the distortion is slightly increased, it is hard to perceive the difference between images coded by the two algorithms, especially at lower rates. The design is implemented with TSMC 0.35-μm 2P4M technology. The area is $1.2 \times 1.2\,\text{mm}^2$, and the simulated clock frequency is 100 MHz. It can process over 4.6 million image pixels within 1 second depending on the required image quality. We have combined it with a DWT module and an 8-bit microprocessor, and the design has been fully verified over an Altera APEX 20K FPGA board. In addition, rate control is easy with SPIHT based methods and no look-up table is essential. Summing up, the design we proposed is certainly attractive for embedded appliances.

References

[1] M. Rabbani and P.W. Hones, Digital Image Compression Techniques, SPIE Opt. Eng. Press, Bellingham, WA, 1991.

[2] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," IEEE Trans. Image Process., vol.1, pp.205–220, April 1992.

[3] M.W. Marcellin, M.J. Gormish, A. Bilgin, and M. Boliek, "An overview of JPEG-2000," Proc. Data Compression Conference, pp.523–541, Snowbird, UT, March 2000.

[4] ISO/IEC 15444-1:2000(E), Information Technology — JPEG 2000 Image Coding System — Part 1: Core Coding System.

[5] A. Said and W.A. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees," IEEE Trans. Circuits Syst. Video Technol., vol.6, no.12, pp.243–250, June 1996.

[6] D. Taubman, "High performance scalable image compression with EBCOT," IEEE Trans. Image Process., vol.9, no.7, pp.1158–1170, July 2000.

[7] J.M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," IEEE Trans. Signal Process., vol.41, no.12, pp.3445–3462, Dec. 1993.

[8] W.B. Pennebaker and J.L. Mitchell, JPEG Still Image Compression Standard, Van Nostrand Reinhold, New York, 1993.

[9] Z. He and S.K. Mitra, "A unified Rate-Distortion analysis framework for transform coding," IEEE Trans. Circuits Syst. Video Technol., vol.11, no.12, pp.1221–1236, Dec. 2001.

[10] C.J. Lian, K.F. Chen, H.H. Chen, and L.G. Chen, "Analysis and architecture design of block-coding engine for EBCOT in JPEG 2000," IEEE Trans. Circuits Syst. Video Technol., vol.13, no.3, pp.219–230, March 2003.

[11] W.B. Huang, Y.J. Chang, A.W.Y. Su, and Y.H. Kuo, "VLSI design of a DWT/modified efficient SPIHT based image codec," International Conference on Information, Communications & Signal Processing — Pacific-Rim Conference on Multimedia (ICICS-PCM) Symposium, Singapore, Dec. 2003.

[12] Synopsys Inc. (2004, Oct. 24). Available:http://www.synopsys.com/cgi-bin/designware/ipdir.cgi?c=DW8051

[13] Center for Image Processing Research, Electrical, Computer, and Systems Engineering Department Rensselaer Polytechnic Institute. Available: http://www.cipr.rpi.edu

[14] F.W. Wheeler and W.A. Pearlman, "SPIHT image compression without lists," IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Istanbul, Turkey, June 2000.

[15] A.W.Y. Su, W.C. Chang, and J.X. Wang, "A new audio compression method based on spectral oriented trees," Proc. Audio Engineering Society (AES) 118th Convention, Paper 6042, Barcelona, Spain, May 2005.

[16] Z. Lu and W.A. Pearlman, "An efficient, low-complexity audio coder delivering multiple levels of quality for interactive applications," IEEE Proc. Multimedia Signal Processing, pp.529–534, Dec. 1998.

[17] M. Raad, A. Mertins, and I. Burnett, "Scalable to lossless audio compression based on perceptual set partitioning in hierarchical trees (PSPIHT)," IEEE Proc. Acoustics, Speech, and Signal Processing, vol.5, pp.v624–627, April 2003.

[18] J. Singh, A. Antoniou, and D.J. Shpak, "Hardware implementation of a wavelet based image compression coder," IEEE Symp. Advances in Digital Filtering and Signal Processing, pp.169–173, 1998.

[19] T.W. Fry and S.A. Hauck, "SPIHT image compression on FPGAs," IEEE Trans. Circuits Syst. Video Technol., vol.15, no.9, pp.1138–1147, Sept. 2005.

[20] Altera Inc. Available: http://www.altera.com/products/devices/apex/overview/apx-overview.html

[21] Xilinx Inc. Available: http://www.xilinx.com/bvdocs/publications/ds022.pdf

[22] Information Technology — JPEG 2000 Image Coding System, ISO/IEC FCD15444-1: 2000 (V1.0, March 2000), p.31.

[23] C. Dunn, "Efficient audio coding with fine-grain scalability," presented at the 111th Convention of the Audio Engineering Society, New York, Nov./Dec. 2001.

[24] M. Raad, A. Mertins, and I. Burnett, "Scalable to lossless audio compression based on perceptual set partitioning in hierarchical trees (PSPIHT)," Proc. IEEE Int. Conf. Acoustics Speech Sig. Proc (ICASSP), 2003.

[25] W.C. Chang, J.X. Wang, and A.W.Y. Su, "Harmonic structure reconstruction in audio compression method based on spectral oriented trees," 120th Convention of Audio Engineering Society, Paper 6809, Paris, France, May 2006.

Appendix A: The Proof of the Rules

Assume that 2^N is the number of width and height of code block. Thus one can use $2N$-bit symbols to represent 2-D addresses in Fig. A· 1. The upper N nibble is for y-axis (height) and the lower N nibble is for x-axis (width). The general form of physical address in a 1-D array is

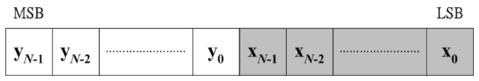

Fig. A· 1 $2N$-bit symbols of an address.

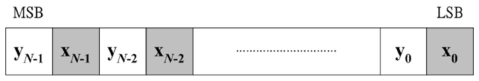

Fig. A· 2 Altering the addressing method.

$$\begin{aligned} &Addr\ (y_{N-1}, y_{N-2}, \ldots, y_0, x_{N-1}, x_{N-2}, \ldots, x_0) \\ &\quad = 2^N \times (y_{N-1} \times 2^{N-1} + y_{N-2} \times 2^{N-2} + \cdots + y_0 \times 2^0) \\ &\qquad + (x_{N-1} \times 2^{N-1} + x_{N-2} \times 2^{N-2} + \cdots + x_0 \times 2^0) \quad \text{(A· 1)} \end{aligned}$$

Therefore, the general form of searching for descendants of the node by using original address description is shown as follows.

$$\begin{aligned} &\textit{Address of direct off-spring No.0} \qquad \text{(A· 2)} \\ &\quad = 2^N \times (y_{N-1} \times 2^{N-1} + \cdots + y_0 \times 2^0) \times 2 \\ &\quad +(x_{N-1} \times 2^{N-1} + \cdots + x_0 \times 2^0) \times 2 \\ &\quad = Addr(y_{N-2}, \ldots, y_0, 0, x_{N-2}, \ldots, x_0, 0) \end{aligned}$$

$$\begin{aligned} &\textit{Address of direct off-spring No.1} \qquad \text{(A· 3)} \\ &\quad = 2^N \times (y_{N-1} \times 2^{N-1} + \cdots + y_0 \times 2^0) \times 2 \\ &\quad +(x_{N-1} \times 2^{N-1} + \cdots + x_0 \times 2^0) \times 2 + 1 \\ &\quad = Addr(y_{N-2}, \ldots, y_0, 0, x_{N-2}, \ldots, x_0, 1) \end{aligned}$$

$$\begin{aligned} &\textit{Address of direct off-spring No.2} \qquad \text{(A· 4)} \\ &\quad = 2^N \times [(y_{N-1} \times 2^{N-1} + \cdots + y_0 \times 2^0) \times 2 + 1] \\ &\quad +(x_{N-1} \times 2^{N-1} + \cdots + x_0 \times 2^0) \times 2 \\ &\quad = Addr(y_{N-2}, \ldots, y_0, 1, x_{N-2}, \ldots, x_0, 0) \end{aligned}$$

$$\begin{aligned} &\textit{Address of direct off-spring No.3} \qquad \text{(A· 5)} \\ &\quad = 2^N \times [(y_{N-1} \times 2^{N-1} + \cdots + y_0 \times 2^0) \times 2 + 1] \\ &\quad +(x_{N-1} \times 2^{N-1} + \cdots + x_0 \times 2^0) \times 2 + 1 \\ &\quad = Addr(y_{N-2}, \ldots, y_0, 1, x_{N-2}, \ldots, x_0, 1) \end{aligned}$$

A new address of the node $(y_{N-1}, y_{N-2}, \ldots, y_0, x_{N-1}, x_{N-2}, \ldots, x_0)$ is modified by altering the addressing method in Fig. A· 2, and is shown as

$$\begin{aligned} &New\ Addr(y_{N-1}, y_{N-2}, \ldots, y_0, x_{N-1}, x_{N-2}, \ldots, x_0) \qquad \text{(A· 6)} \\ &\quad = Addr(y_{N-1}, x_{N-1}, \ldots, y_{N/2}, x_{N/2}, y_{N/2-1}, x_{N/2-1} \ldots, \\ &\qquad y_0, x_0) = 2^N \times (y_{N-1} \times 2^{N-1} + x_{N-1} \times 2^{N-2} + \cdots \\ &\qquad + y_{N/2} \times 2^1 + x_{N/2} \times 2^0) + (y_{N/2-1} \times 2^{N-1} \\ &\qquad + x_{N/2-1} \times 2^{N-2} + \cdots + y_0 \times 2^1 + x_0 \times 2^0) \end{aligned}$$

Therefore, the addressing rule of the direct off-springs of a coefficient is modified as follows.

$$\begin{aligned} &\textit{New Address of direct off-spring No.0} \qquad \text{(A· 7)} \\ &\quad = NewAddr(y_{N-2}, \ldots, y_0, 0, x_{N-2}, \ldots, x_0, 0) \end{aligned}$$

$$= 2^N \times (y_{N-2} \times 2^{N-1} + x_{N-2} \times 2^{N-2} + \cdots + x_{N/2-1} \times 2^0)$$
$$+(y_{N/2-2} \times 2^{N-1} + x_{N/2-2} \times 2^{N-2} + \cdots$$
$$+y_0 \times 2^3 + x_0 \times 2^2)$$
$$= NewAddr(y_{N-1}, \ldots, y_0, x_{N-1}, \ldots, x_0) \times 2^2$$
$$= \textit{New address of parent node} \times 2^2$$

New Address of direct off-spring No.1 (A· 8)

$$= NewAddr(y_{N-2}, \ldots, y_0, 0, x_{N-2}, \ldots, x_0, 1)$$
$$= NewAddr(y_{N-1}, \ldots, y_0, x_{N-1}, \ldots, x_0) \times 2^2 + 1$$
$$= \textit{New address of parent node} \times 2^2 + 1$$

New Address of direct off-spring No.2 (A· 9)

$$= NewAddr(y_{N-2}, \ldots, y_0, 1, x_{N-2}, \ldots, x_0, 0)$$
$$= NewAddr(y_{N-1}, \ldots, y_0, x_{N-1}, \ldots, x_0) \times 2^2 + 2$$
$$= \textit{New address of parent node} \times 2^2 + 2$$

New Address of direct off-spring No.3 (A· 10)

$$= NewAddr(y_{N-2}, \ldots, y_0, 1, x_{N-2}, \ldots, x_0, 1)$$
$$= NewAddr(y_{N-1}, \ldots, y_0, x_{N-1}, \ldots, x_0) \times 2^2 + 3$$
$$= \textit{New address of parent node} \times 2^2 + 3$$

Because the rule in Eqs. (A· 2)–(A· 6) is always true except at the highest and lowest pyramid levels [5]. The rule in Eqs. (A· 7)–(A· 10) also does not change by the new addressing method for the square 2^N processed block.

Appendix B: The Descirption of the FSM

Referring to Fig. 10, the meaning of each state is represented as follows:

1. Idle state: Encode_CORE do nothing and wait for the start signal from the Encode_Controller module.
2. The SearchMax state: Set the initial value of LIP, LIS and LSP flag and find out the maximal coefficient in this code block.
3. The CaculateN0 state: Prevent the maximal coefficient from being zero.
4. The CaculateN1 state: Find out $n = \lfloor \log_2(max\{| c_i |\}) \rfloor$, $0 \leq i \leq N$.
5. The OutputN state: Output the value n into the local memory and set the threshold.
6. The RefinementPass state: Check if the current node has been significant and update the threshold. Set the LSP flag of current node being 1 when the LSP flag of current node is 2.
7. The SortingPass0 state: If the current node is insignificant, output the $S_n(i)$. If the coefficient of the insignificant node is greater than threshold, the LSP flag and LIP flag of the current node is reset and set *MaxLSP* being the address of the current node.
8. The SortingPass1 state: Output the sign of the current node.
9. The SortingPass2 state: Find out the first and the last address of the offspring of the current node.
10. The CheckD state: Output the result when the current node is greater than threshold or the search for all descendants is finished. Set the LIS flag being type B when address of the current node is smaller than the last node, otherwise set the LIS flag being 0.
11. The SortingPass3 state: Output the compared result between the current node and threshold. Besides set the LSP flag being 2 and *MaxLSP* being the address of the current node when the coefficient is greater than threshold, otherwise set the LSP flag being 1 and *MaxLIP* being the address of the current node.
12. The SortingPass4 state: Output the sign of the current node.
13. The SortingPass5 state: Find out the first and the last address of the offspring of the current node.
14. The CheckL state: Output the result when the current node is greater than threshold or the search for all descendants is finished. Set the LIS flag being 0 when address of the current node is greater than the last node.
15. The SortingPass6 state: Set the LIS flag being type A and *MaxLIS* being the maximal address of these descendants of the current node.
16. The SortingPass7 state: Update the address of the next searching for the insignificant nodes.

Win-Bin Huang was born in Kaohsiung, Taiwan, in 1977. Her received the B.S. degree in mathematics and the M.S. degree in computer science information engineering from National Cheng-Kung University, Taiwan, in 2000 and 2002, respectively. He is currently working for the Ph.D. degree in Computer Science and Information Engineering from National Cheng Kung University, Taiwan. His research activities include Digital Signal Processing, VLSI Design Technology and CAD, Neural Network, Information Security, Image/Video signal processing.

Alvin W.Y. Su was born in Taiwan, Taiwan, in 1964. He received the B.S. degree in control engineering from National Chiao-Tung University, Hsinchu, Taiwan, in 1986, and the M.S. and Ph.D. degrees in electrical engineering from Polytechnic University, Brooklyn, NY, in 1990 and 1993, respectively. From 1993 to 1994, he was with the Center for Computer Research in Music and Acoustics (CCRMA), Stanford University, Stanford, CA. From 1994 to 1995, he was with Computer Communication Laboratory, Industrial Technology Research Institute, Taiwan. In 1995, he joined the Department of Information Engineering and Computer Engineering, Chung-Hwa University, Taiwan, where he serves as an Associate Professor. He joined the Department of Computer Science and Information Engineering, National Cheng-Kung University, Tainan, Taiwan, in 2000. His research interests cover the areas of digital audio signal processing, physical modeling of acoustic musical instruments, human computer interface design, pattern recognition, data compression, image/video signal processing, and VLSI signal processing.

Yau-Hwang Kuo was born in Tainan, Taiwan in 1959. He received M.S. and Ph.D. degrees in Computer Engineering from National Cheng Kung University in 1984 and 1988. He was the President of Taiwanese AI Association from 1999 to 2000, the Director of Research Center for Computer System Technology from 1997 to 2000, the Managing Director of Chinese Fuzzy System Association from 1996 to 2000, the Director of Center for Research of E-life Digital Technology, and the coordinator of Computer Science and Information Engineering Program of National Science Council, from 2002 to 2005. He is currently Professor with the Department of Computer Science and Information Engineering, National Cheng Kung University. He is also the chairman of Computer Center, Ministry of Education, R.O.C. His research interests include intelligent computing, knowledge management, broadband communication, information retrieval, pattern recognition, and VLSI design.