International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 11, November-2014

Hardware Implementation of Tag Tree in JPEG2000 Encoder

Bach Tuan Dong
Department of Electrical and Electronic Engineering
Ho Chi Minh City University of Technology and Education
Ho Chi Minh City, Vietnam

Trinh Hoai An
Department of Electrical and Electronic Engineering
Ho Chi Minh City University of Technology and Education
Ho Chi Minh City, Vietnam

Huynh Hong Tram
Department of Computational Engineering
Vietnamese – German University
Binh Duong Province, Vietnam

Le My Ha
Department of Electrical and Electronic Engineering
Ho Chi Minh City University of Technology and Education
Ho Chi Minh City, Vietnam

***Abstract*—Hierarchical data structure takes an importance rule in many files such as computer graphic, information storage and retrieval, image processing etc. Quad-tree is a kind of hierarchical data structure which is applied widely in image coding. Tag tree, a particular type of quad-tree, is a coding mechanism used in tier-2 coding engine in JPEG2000 encode. In this paper, we propose a hardware implementation of tag tree in JPEG2000 encode system. Tag tree is successful implemented on hardware system, tested by using OpenJPEG and simulated on Modelsim and VCS tools.**

***Keywords*—Quad-tree; tag tree; JPEG2000; OpenJPEG; packet; packet header; node; parent node**

I. INTRODUCTION

JPEG2000 is a new image compression standard, which was created by Joint Photographic Expert Group committed in 2000. It shows many advances such as superior compression performance, multiple resolution representation, progressive transition support, lossless and lossy compression etc. Based on the independent coding mechanism, with JPEG2000, the relevant byte in a compressed codestream can be extracted and reassembled into a different codestream without decompress the codestream [2]. The only thing we have to do is changing the packing way of the codestream, which is processed in Tier 2. Tier-2 reorders and packs the codeblock bit-stream into the full-feature bit-stream.

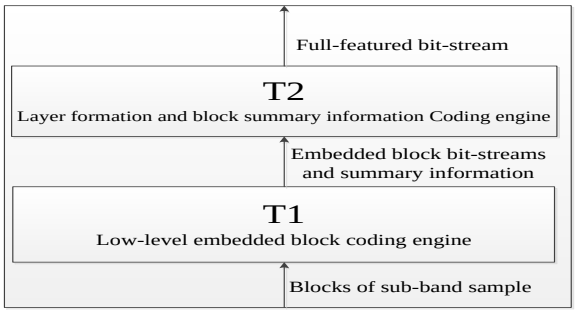

Fig. 1. Tier-1 coding and Tier-2 coding in JPEG2000

All compressed image data representing a specific tile, layer, component, resolution level and precinct appears in the code stream in a contiguous segment called a packet. Packet data is aligned at 8-bit (one byte) boundaries [1].

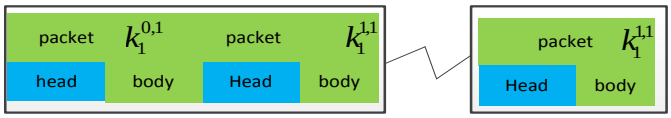

Fig. 2. Bit-stream organization

A packet includes packet header and packet data, which is described as follows.

A packet header describes:

- Zero length packet;
- Code-block inclusion;
- Zero bit-plane information;
- Number of coding passes;
- Length of the code-block compressed image data from a given code-block.

The code-block inclusion and zero bit-plane information are coded with the tag tree coding algorithm.

This paper is organized as follows. The section 2 deals with the introduction of tag tree coding scheme. Hardware implementation of tag tree is discussed in section 3. Finally, the result and conclusion are presented in section 4 and section 5.

II. TAG TREE ENCODING PROCEDURE

A. *Tag tree encode algorithm*

Tag tree is a way of representing a two-dimensional array of non-negative integer in hierarchical way. It successive creates reduced resolution levels of this two dimension array, forming a tree. At every node of this tree the minimum integer of the (up to four) nodes below it is recorded [1].

The following is an example of a tag tree. Assuming that $q_i(m,n)$ is the value of an array at row m, column n and level i.

1 $q_3(0,0)$	3 $q_3(0,1)$	2 $q_3(0,2)$	3 $q_3(0,3)$	2 $q_3(0,4)$	3 $q_3(0,5)$
2 $q_3(1,0)$	2 $q_3(1,1)$	1 $q_3(1,2)$	4 $q_3(1,3)$	3 $q_3(1,4)$	2 $q_3(1,5)$
2 $q_3(2,0)$	2 $q_3(2,1)$	2 $q_3(2,2)$	2 $q_3(2,3)$	1 $q_3(2,4)$	2 $q_3(2,5)$

Fig. 3. Level 3 of tag tree

$Min(1,3,2,2)-1$ $q_2(0,0)$	$Min(2,3,1,4)-1$ $q_2(0,1)$	$Min(2,3,3,2)-2$ $q_2(0,2)$
$Min(2,2)-2$ $q_2(1,0)$	$Min(2,2)-2$ $q_2(1,1)$	$Min(1,2)-1$ $q_2(1,2)$

Fig. 4. Level 2 of tag tree

1 $q_1(0,0)$	1 $q_1(0,1)$

Fig. 5. Level 1 of tag tree

$q_0(0,0)$

Fig. 6. Level 0 of tag tree

B. Encoding method:

Each coded node of a tag tree includes d bit zeros and one bit 1 (d is the difference of the value of the current node and its parent node). Specially, the parent node of a root node is 0.

1) Example 1:

Encode $q_3(0,0)$ in the previous example. In order to encode the node $q_3(0,0)$, we should consider $q_2(0,0)$, $q_1(0,0)$, $q_0(0,0)$

The parent node of $q_0(0,0)$ is 0; and $q_0(0,0) = 1 \rightarrow q_0(0,0) - 0 = 1 \rightarrow$ the coding value at the node $q_0(0,0)$ includes one bit 0 and one bit 1 $\rightarrow$ the coding value of $q_3(0,0)$ is "01".

The parent node of $q_1(0,0)$ is $q_0(0,0)$; and $q_1(0,0) = 1$, $q_0(0,0) = 1 \rightarrow q_1(0,0) - q_0(0,0) = 0 \rightarrow$ the coding value of $q_1(0,0)$ includes zero bit 0 and one bit 1. Combining with the coding value of the previous node, the current coding value is "0 1 1".

Similarly, the parent node of $q_2(0,0)$ is $q_1(0,0)$; and $q_2(0,0) = 1$ and $q_1(0,0) = 1 \rightarrow q_2(0,0) - q_1(0,0) = 0 \rightarrow$ the coding value at the node $q_2(0,0)$ includes zero bit 0 and one bit 1. Combining with the coding value of the previous nodes, the current coding value is "0 1 1 1".

Similarly, the coding value at the node $q_3(0,0)$ is "0 1 1 1 1"

2) Example 2:

Coding $q_3(0,1)$. The parent node of $q_3(0,1)$ is $q_2(0,0)$. Because the node $q_2(0,0)$ has already coded, we just consider $q_2(0,0)$ when coding $q_3(0,1)$.

The parent node of $q_3(0,1)$ is $q_2(0,0)$, $q_3(0,1) - q_2(0,0) = 2 \rightarrow$ the coding value of the node $q_3(0,1)$ includes two bit 0s and one bit 1 $\rightarrow$ The coding value is "001".

III. IMPLEMENT TAG TREE ON HARDWARE SYSTEM

A. Some problems had to solve to build a tag tree on hardware system

- In order to encode data at a specific position, the tree structure has to build first (for example, the coding value of $q_3(0,0)$ is obtained by finding the differences of $q_0(0,0)$ and 0, $q_1(0,0)$ and $q_0(0,0)$, $q_2(0,0)$ and $q_1(0,0)$, $q_3(0,0)$ and $q_2(0,0)$
- To determine whether a node is encoded, a flag is used to mark an encoded node. If that node is not encoded, encode that node and jump to its parent node. If that node is encoded, encode that not and stop the coding process for that node.
- Beginning from the input array of the maximum level, we can find the minimum values of groups of four elements to determine array data values of the next levels.
- The number of coding bit of a tag tree cannot be predetermined; it depends on the input values. Therefore, we need two kinds of signals to express the output. The first one is the "coding", which shows the coded values. The second one is "mask", which shows the number of valid bits in the "coding" signal.

The tag tree on hardware system is divided into 3 parts:

- Part 1: Save data into RAM and build the tree.
- Part 2: Read data from RAM and encode.
- Part 3: Pack encoded data into packets and send packet out.

B. Tag tree hardware architecture overview

1) Part 1:

The input data is sent to the tag tree module in raster order. "Line buffer" module captures data and groups them into groups of four elements. Each group is sent to "min" module, which finds the minimum value of groups of four elements. These minimum values are saved in RAM. Besides, these minimum values also push out to the next level. The similar method is applied for all levels.

The tree structure is generated and saved in RAM.

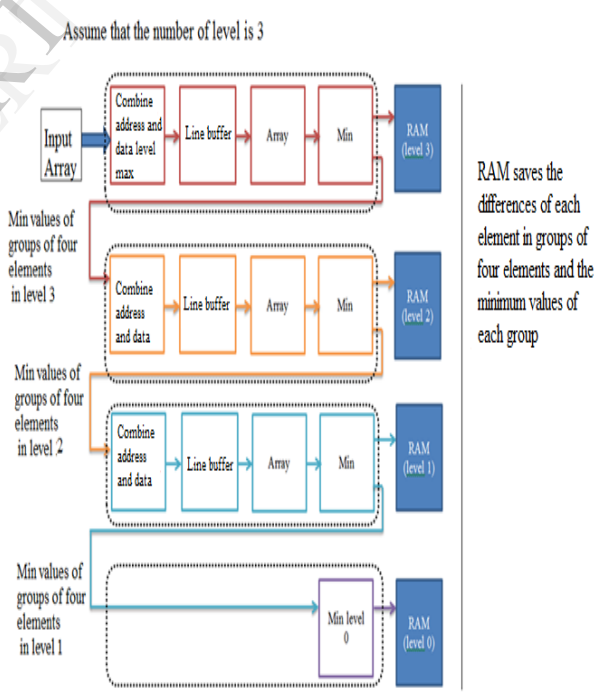

Fig. 7. Tag tree structure - Part 1

2) Part 2:

Part 2 is the reading (from RAM) and encoding process. The main purpose of this part is sending out the coding values which are suitable with the input positions.

3) Part 3:

Part 3 is the parking process. It sends out packets and mask packets.

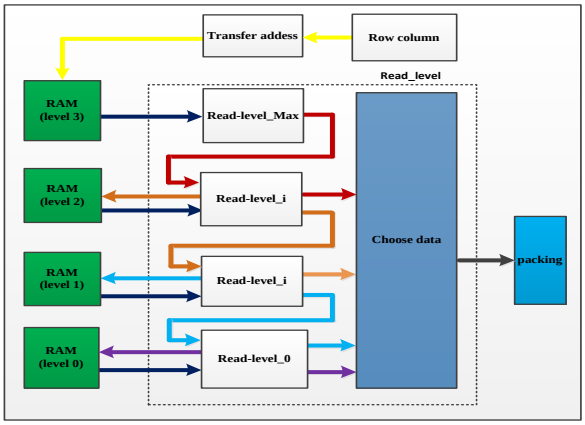

Fig. 8. Tag tree structure - Part 2 and 3

C. *Tag tree architecture in details*

1) Module "combine address and data"

Combine data_in , row_in and column _in into a data line.

2) Module "line buffer"

a) Function:

Create two consecutive rows.

b) Example:

Assume that we have a 4x4-input array. The output data from the "line buffer" module is described as following:

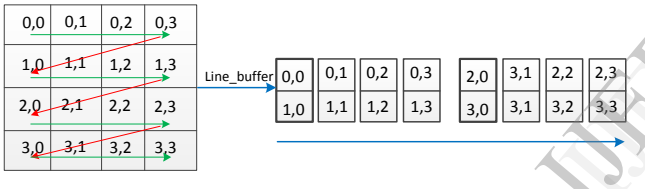

Fig. 9. Line buffer – even height

In case of the height is odd, the values at A, B, C, D is the maximum performed value. For example, we need 4 bits to perform the input values of "line buffer"; it means that the values at cells A, B, C, D are 1111_b.

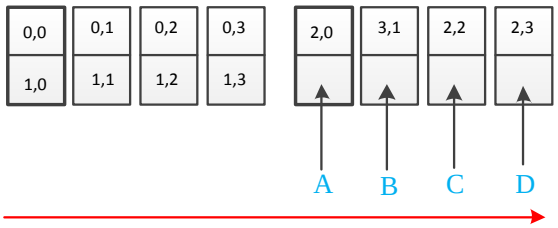

Fig. 10. Line Buffer –odd height

The output values of the "line buffer" module are the input values of the "array" module.

3) Module "Array"

Group the data out from the "line buffer" module into groups of four elements. The input value gotten from the "line buffer" module is processed by the "array" module and creates the following output order:

Fig. 11. Array Module

Each group includes 4 data lines sent to the "min" module in parallel.

The main purpose of "combine_address_data", "line buffer" and "array" is grouping data into arrays of 2x2 elements and inserting the maximum possible values into every missing element. Therefore, there are always 4 data lines sent to the "min" module.

4) Module "min":

Find the minimum value of 4 elements gotten from "array" module.

Create the information described for a node. The information includes parent node address, flag (which is used to mark coded nodes) and the differences of each element in a 2x2 array and the minimum value of that 2x2 array. The structure of a tag tree is saved in RAM.

5) Save tag tree module into RAM

Consider the following array as an example:

Level = 3, $width_3 = 6$, $height_3 = 2$;

1 $q_3(0,0)$	3 $q_3(0,1)$	2 $q_3(0,2)$	3 $q_3(0,3)$	2 $q_3(0,4)$	3 $q_3(0,5)$
2 $q_3(1,0)$	2 $q_3(1,1)$	1 $q_3(1,2)$	4 $q_3(1,3)$	3 $q_3(1,4)$	2 $q_3(1,5)$
2 $q_3(2,0)$	2 $q_3(2,1)$	2 $q_3(2,2)$	2 $q_3(2,3)$	1 $q_3(2,4)$	2 $q_3(2,5)$

Fig. 12. Group data in level 3

Consider the 2x2 green array; min ($q_3(0,0)$, $q_3(0,1)$, $q_3(1,0)$, $q_3(1,1)$) = $q_2(0,0)$, $q_2(0,0)$ (the 1x1 yellow array) is the parent node of the green array.

Level = 2; $width_2 = 3$; $height_2 = 2$

$Min(1,3,2,2)-1$ $q_2(0,0)$	$Min(2,3,1,4)-1$ $q_2(0,1)$	$Min(2,3,3,2)-2$ $q_2(0,2)$
$Min(2,2)-2$ $q_2(1,0)$	$Min(2,2)-2$ $q_2(1,1)$	$Min(1,2)-1$ $q_2(1,2)$

Fig. 13. Group data in level 2

Level = 1; $width_1 = 2$; $height_1 = 1$

1 $q_1(0,0)$	1 $q_1(0,1)$

Fig. 84. Group data in level 2

Each cell in RAM describes the information of a 2x2 array block. It means that an address in RAM permits to access to a 2x2-block. Following with an address line is a position line, which permits to access a specific value in that 2x2-block.

6) Module "Transfer address"

The input node address is described in term of row and column, the "transfer address" module gets that row and column and changes them into respective address to access RAM level max (address and position)

This module is used once in case of level max, the remaining levels of RAM base on the parent node saved in RAM at that level to access the upper level.

7) Module "Read_level":

The encoding is proceeded here. It is connected to RAM in encode period to control reading data out from RAM and writing the flag when finishing coding a node.

The output data of this module is "coding" and "mask". "Coding" is the encoded value, and "mask" is the number of valid bit in "coding".

The "read_level" module includes 2 parts: "coding" and "choose". "Coding" encodes for a node, "choose" selects one data line of levels to send out.

The following is the data flow of the "read_level" module.

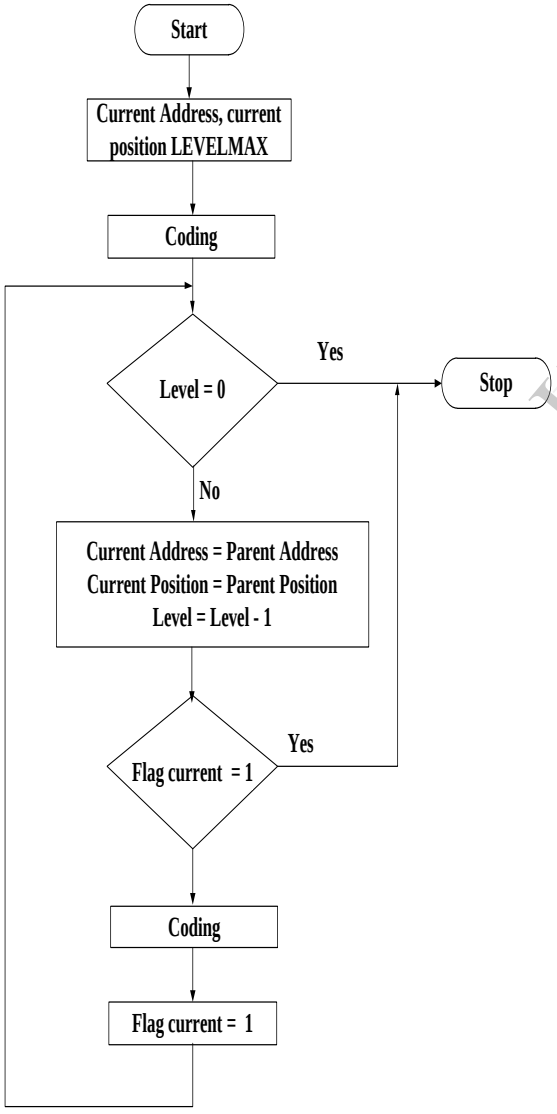

Fig. 95. Encoding flow of the "read_level" module.

IV. RESULT

A. Parameter

- *Level max =* $max([\log_2(WITH\_TAG\_TREE)],[\log_2(HEIGHT\_TAG\_TREE)])$
- *Memlength(i)=* $\sum_{0}^{Levelmax} \{[\left(\frac{WIDTHi}{2}\right)]*[\left(\frac{HEIGHTi}{2}\right)]\}$
- WIDTH_LEVEL= $\sum_{level}^{Levelmax} [\left(\frac{WIDTHlevel}{2}\right)]$
- HEIGHT_LEVEL= $\sum_{level}^{Levelmax} [\left(\frac{HEIGHTlevel}{2}\right)]$

B. Compile result

Assume that WIDTH = 32, HEIGHT = 32, PACKET_LEN = 8, we have the compile result on Quartus II version 11.0:

Altera Device	Area	Fmax (MHz)	Mem	DSP
Cyclone II EP2C35F672C6	14,867 LEs	92.89	0	0
Stratix II EP2S30F672C3	4,891 ALUTs	151.38	0	0

V. CONCLUSION

Based on the tag tree encoding algorithm, this paper presents the implementation of a tag tree on hardware system. This core is processed through three parts. Part 1 is saving data and building the tree structure. Part 2 is encoding; and part 3 is packing. Although the input array values of a tag tree require the raster-scan order, the information in the tree may be coded in any order. The input array dimension, data input bit-width and the maximum length of a packet are defined in an inclusion file by users. The architecture is successfully applied on theJPEG2000 Encoder IP of ICDREC. The tag tree operates at more than 150 MHz on Stratix II Family of Altera FPGA

REFERENCES

[1] JPEG2000 Part 1 Final Committee Draft Version 1.0, ISO/IEC JTC1/SC 29/WG N1646R, 2000.
[2] David S. Taubman, JPEG2000 Image Compression Fundamentals, Standards and Practice, 2002
[3] Coding of Still Pictures, ISO/IEC JTC 1/SC 29/WG 1 WG1N1684, April 25, 2000
[4] Schelkens, Skodras, Ebrahimi *The JPEG 2000 Suite*, 2009
[5] Chung-JrLian, "Lifting Based Discrete Wavelet Transform Architecture for JPEG2000", 2009