SAJET

South Asian Journal of Engineering and Technology Vol.2, No.18 (2016) 1–11

ISSN No: 2454-9614

Implementation of Arithmetic Coder Used in SPIHT Algorithm

S. Karthik ,V. Annapoorani*

Department Of Electronics and Communication Engineering, Mahendra Institute of Technology
Mallasamudram, Namakkal, India

*Corresponding Author: S. Karthik

E-mail: karthikss42@gmail.com

Received: 02/11/2015, Revised: 10/11/2015 and Accepted: 13/03/2016

Abstract

In this paper Set Partitioning in Hierarchical Trees (SPIHT) algorithm for image compression is proposed with a arithmetic coder thereby it compresses the Discrete Wavelet Transform decomposed images. This architecture is advantageous from various optimizations performed at different levels of arithmetic coding from higher algorithm abstraction to lower circuit implementation. SPIHT has straightforward coding procedure and requires no tables which make a SPIHT algorithm an appropriate one for low cost hardware implementation. In order to avoid rescanning the wavelet transformed coefficients a breadth first search SPIHT without lists is used instead of SPIHT with lists. With the help of Breadth First search high speed architecture is achieved. Dedicated circuit such as common bit detector is used for loop unrolling the renormalization stage of arithmetic coding. Critical path in the architecture are shortened by employing Floating point multiplier and carry look ahead adder. Design has been implemented on Spartan 6 FPGA.

Keywords: Arithmetic coding, Common bit detection (CBD) circuit, Discrete wavelet transform (DWT), Set Partitioning in Hierarchical Trees (SPIHT). **Reviewed by* **ICETSET'16** *organizing committee*

1. Introduction

Image compression is an application of image processing performed on digital images. The main objective of image compression is to reduce redundancy of the image data in order to be able to store or transmit data in an efficient form. Uncompressed multimedia (audio and video) data requires considerable storage capacity and transmission band width. **[2]** The main goal of compression is to provide sufficient storage space, wider transmission bandwidth and a longer transmission time for image, audio and video data.

At present state of technology, the only solution to compress multimedia data before its storage and transmission and decompress it at the receiver. A Common characteristic of images is that the neighboring pixels are correlated and therefore contain redundant information. Two fundamental components of compression are redundancy and irrelevancy reduction. Redundancy reduction aims at removing duplication from the signal source

(image/video).

Irrelevance reduction omits only those parts of the signal that are not noticed by the signal receiver. In general, three types of redundancies that can be identified namely: Spatial Redundancy or correlation between neighboring pixel values. Spectral redundancy or correlation between different color-planes or spectral-bands. Temporal redundancy or correlation between adjacent frames in a sequence of images especially in video applications. Compression techniques are classified into two namely lossy compression and lossless compression.

Lossless compression technique guarantees the exact duplication of input in compress/decompress cycle. This is mainly used in medical applications. Methods for lossless image compression are: Run-length encoding, DPCM and Predictive Coding and Entropy encoding. With Lossy compression technique higher levels of data reduction is possible but it results in less perfect reproduction of the original image. This is useful in applications such as broadcast television, video conferencing, and facsimile transmission, in which a certain amount of error is acceptable trade-off for increased compression performance. Methods for lossy image compression are: Reducing the color space to the most common colors in the image, chroma sub-sampling, transform coding, and fractal compression. **[1]** Generally a compression system consists of encoder and decoder stages. Encoder consists of the mapper, quantizer and encoder. Mapper transforms the input image into a format by reducing the inter-pixel redundancies. Quantizing refers to a reduction of the precision of the floating point values of the wavelet transform. A symbol encoder further compresses the quantized values to give a better overall compression.

It uses a model to accurately determine the probabilities for each quantized value and produces an appropriate code based on these probabilities so that the resultant output code stream will be smaller than the input stream. An arithmetic coder or Huffman coder is used as symbol encoders. Decoder part of the compression model consists of symbol decoder and inverse mapper. Among these FPGAs are best suited to implement image processing algorithms because they offers features like infinite reprogram ability, high parallelism, flexibility, computational power and short development time**[3,4].**

1.1. Discrete Wavelet Transform

A Discrete Wavelet Transform is a wavelet transform in which wavelets are discretely sampled. If Discrete Wavelet Transform is compared with the Fourier Transform it has an advantage of temporal resolution which captures both frequency and location information. A wave is an oscillating function of time or space and is periodic. In contrast, wavelets are localized waves. Wavelets have their energy concentrated in time and they are used for analysis of transient signals.

Wavelet transform uses wavelets of finite energy whereas the Fourier Transform and short Time Fourier Transform uses wavelets of finite energy. A line based Wavelet Lifting scheme is used which mainy consists of three stages namely Split, Predict and Update Stages respectively. Figure 2.1 shows the diagram for Forward transform for lifting scheme**.[6]**

i)Split

This step is often referred to as Lazy Wavelet transform. In this stage the input pixel values obtained from Input image „X is divided into even samples Si+1=Xe={x2j} and odd samples Di+1=Xo={x2j+1}, where „i represents the element and „j represents the iteration.

ii) Predict:

This step is often referred to as dual lifting. In this stage the odd elements of the next iteration are predicted from the even samples of the present iteration. Where „i- represents the element and „j"-represents the iteration.

Di+1=Di+1-P(Si+1) (1)

ii) Update:

This step is referred to as primary lifting. In this stage the even elements of next iteration are calculated from the odd samples of the update stage.Update stage follows the predict stage.

2. Algorithm

2.1. Set Partitioning In Hierarchical Trees Algorithm

SPIHT is an image compression algorithm with lists to store the significant information of wavelet coefficients for n image coding purpose. It mainly uses three different lists namely List of Significant Pixels(LSP),List of Insignificant sets (LIS) and list of Insignificant pixels(LIP).In SPIHT algorithm initially all wavelet coefficients are n considered as insignificant sets and these sets are placed in the list called LIS. Wavelet coefficients in LIS are compared with a predefined threshold if the wavelet coefficient value is greater than the predefined threshold then they are placed in the list called LSP and if the value is lesser than the predefined threshold then the wavelet coefficients are placed in the list called LIP. All the pixel values in the list LIS are tested for there significant state. Significant state of the wavelet coefficients are tested using the equation (3) which is as given below.So, this reason why SPIHT algorithm less= bits to code after the discrete wavelet transform. **[4]**

2.2. SPIHT encoding

The input image is gray scale image of size 512x512 each pixel of 8 bits. Input image is converted to pixel values in MATLAB. Pixel values obtained are given as input to the Line based wavelet lifting module which produces wavelet coefficients. The transformed wavelet coefficients are placed in a buffer and they are accessed in a SPIHT-Breadth First Search way. Processor dispatcher dispatches the transformed wavelet coefficients to the arithmetic coder through internal bus. Output of arithmetic coder is provided to the internal bus and sent to the code FIFO. The Read FIFO and Truncate module are responsible for the final code stream formation, which reads each code FIFO from top to bottom and truncates the code stream according to the bit rate requirement.

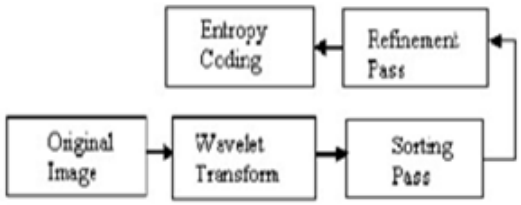

Fig. 1 Algorithm for SHIPT Encoding

3. Literature Survey

3.1. Line Based Lifting Wavelet Engine

Lifting step scheme consists of three simple phases: the first step, or Lazy wavelet, splits the data into two subsets: even and odd; the second step calculates the wavelet coefficients (high-pass) as the failure to predict the odd set based on the even set; finally the third step updates the even set using the wavelet coefficients to compute the scaling function coefficients (low-pass). The predict phase ensures polynomial cancellation in the high-pass, and the update phase ensures preservation of moments in the low-pass.In folded architecture the output of processing element is fed back through the delay registers. By adding different n number of delay registers and coefficients of processing elements folded architecture based lifting scheme can be used to design different wavelets reduced. **[6]** In folded architecture the output of processing element is fed back through the delay registers. By adding different number of delay registers and coefficients of processing elements folded architecture based lifting scheme can be used to design different wavelets.

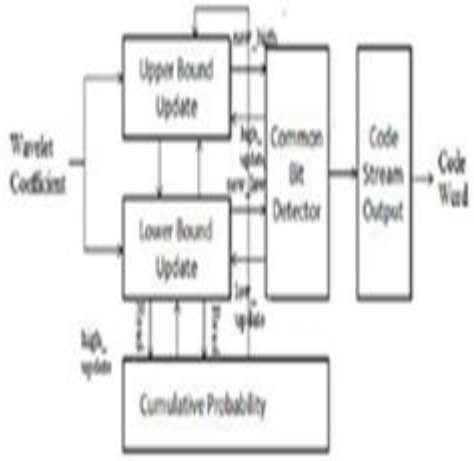

Fig. 2 Architecture of Arithmetic Coder

Significant wavelet coefficients are provided as input to the arithmetic coder. High and low values are multiplied with cumulative probabilities and updated high and low values are given as input to common bit detector unit. Output is then passed to the code stream output unit in arithmetic coder. Cumulative probability part will hold

the frequency of occurrence of wavelet coefficient.

3.2. Upper Bound and Lower Bound Update

Upper bound and lower bound update registers consists of Look ahead adder and Fast array multiplier. The calculation units for upper and lower bound update units are as shown.

Output of upper and lower update units are new_low and new_high. For speed up purpose, a Carry Look Ahead adder and a floating point multiplier are employed to reduce the delay of critical path. The cumulative probabilities cum_freq[i] and cum_freq [i-1] are provided by the cumulative probability module and this is accompanied by high and new values to compute new bounds for the probability interval these are calculated.

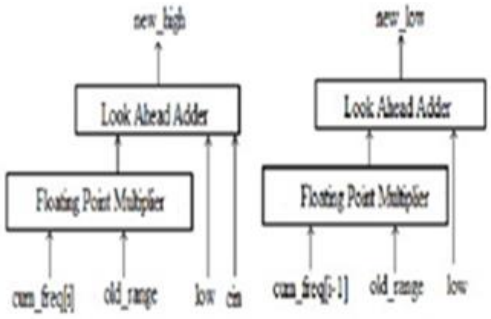

Fig. 3 Upper Bound and Lower Bound Limit

4. Tests and Results

The design of SPIHT algorithm with arithmetic coder is described in VHDL. Design and has been carried out. Figure 4.1 depicts the simulation results of lifting based DWT. The input to this module is the pixel values stored in 1-D array. Output of this module is a transformed coefficient. Output of DWT is provided as input to SPIHT algorithm so the transformed wavelet coefficients are initially stored in the List of Insignificant Sets (LIS) now the wavelet coefficients in the LIS are compared with the threshold value if the wavelet coefficients are greater than the threshold then they are placed in List of significant pixels otherwise they are placed in List of Insignificant Pixels.

The challenges for designers to solve during real-time implementation. First, a large amount of coding symbols is supplied to arithmetic coder which can be a bottleneck for high speed real-time applications. Because the scheme of SPIHT is a bit-plane based method, which codes each bit-plane from the most significant bit-plane (MSB) to the least significant bit-plane (LSB) sequentially, the quantity of context symbols for arithmetic coder will be proportional to the coded planes that are determined by the maximal wavelet coefficient.

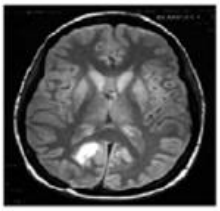

Fig. 4 Input Image

Fig. 5 2DWT Image

4.1. Existing system

The power for Higher update block is shown in below Fig.

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		42
Vccint 1.80V:	20	36
Vcco33 3.30V:	2	7
Clocks:	4	8
Inputs:	0	1
Logic:	0	0
Outputs:		

Fig. 6 Higher update block power

The Area for Higher update block is shown in below Fig.

```
Mapped Date    : Thu May 22 18:46:02 2014

Design Summary
--------------
Number of errors:      0
Number of warnings:   14
Logic Utilization:
  Number of Slice Latches:          102 out of 13,824    1%
  Number of 4 input LUTs:           252 out of 13,824    1%
Logic Distribution:
    Number of occupied Slices:                          145 out of  6,912    2%
    Number of Slices containing only related logic:     145 out of    145  100%
    Number of Slices containing unrelated logic:          0 out of    145    0%
        *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:          268 out of 13,824    1%
      Number used as logic:                     252
      Number used as a route-thru:               16
   Number of bonded IOBs:            48 out of    325   14%
      IOB Latches:                                 2
   Number of GCLKs:                  1 out of      4   25%
   Number of GCLKIOBs:               1 out of      4   25%

Total equivalent gate count for design:  2,849
Additional JTAG gate count for IOBs:  2,352
Peak Memory Usage:  137 MB

NOTES:

   Related logic is defined as being logic that shares connectivity - e.g. two
   LUTs are "related" if they share common inputs.  When assembling slices,
```

Fig. 7 Higher update block Area

The Area for Lower update block is shown in below Fig.

```
Number of warnings:   14
Logic Utilization:
  Number of Slice Latches:           51 out of 13,824    1%
  Number of 4 input LUTs:           185 out of 13,824    1%
Logic Distribution:
    Number of occupied Slices:                          109 out of  6,912    1%
    Number of Slices containing only related logic:     109 out of    109  100%
    Number of Slices containing unrelated logic:          0 out of    109    0%
        *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:          201 out of 13,824    1%
      Number used as logic:                     185
      Number used as a route-thru:               16
   Number of bonded IOBs:            48 out of    325   14%
      IOB Latches:                                 2
   Number of GCLKs:                  1 out of      4   25%
   Number of GCLKIOBs:               1 out of      4   25%

Total equivalent gate count for design:  2,192
Additional JTAG gate count for IOBs:  2,352
Peak Memory Usage:  137 MB

NOTES:

   Related logic is defined as being logic that shares connectivity - e.g. two
   LUTs are "related" if they share common inputs.  When assembling slices,
   Map gives priority to combine logic that is related.  Doing so results in
   the best timing performance.

   Unrelated logic shares no connectivity.  Map will only begin packing
   unrelated logic into a slice once 99% of the slices are occupied through
```

Fig. 8 Lower update block Area

The power for Lower update block is shown in below Fig.

XPower and Datasheet may have some Quiescent Current differences. This is due to the fact that the quiescent numbers in XPower are based on measurements of real designs with active functional elements reflecting real world design scenarios.

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		40
Vccint 1.80V:	19	34
Vcco33 3.30V:	2	7
Clocks:	3	6
Inputs:	0	1

Fig. 9 Lower update block Power

4.2. Proposed system

The Area for Higher update block is shown in below Fig.

```
Mapper Version : spartan2e -- $Revision: 1.34 $
Mapped Date    : Thu May 22 18:53:38 2014

Design Summary
--------------
Number of errors:      0
Number of warnings:   14
Logic Utilization:
  Number of Slice Latches:          45 out of 13,824    1%
  Number of 4 input LUTs:          204 out of 13,824    1%
Logic Distribution:
    Number of occupied Slices:                        120 out of  6,912    1%
    Number of Slices containing only related logic:   120 out of    120  100%
    Number of Slices containing unrelated logic:        0 out of    120    0%
        *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:         220 out of 13,824    1%
      Number used as logic:                    204
      Number used as a route-thru:              16
   Number of bonded IOBs:           48 out of    325   14%
      IOB Latches:                               2
   Number of GCLKs:                 1 out of      4   25%
   Number of GCLKIOBs:              1 out of      4   25%

Total equivalent gate count for design:  2,267
Additional JTAG gate count for IOBs:  2,352
Peak Memory Usage:  137 MB

NOTES:

   Related logic is defined as being logic that shares connectivity - e.g. two
```

Fig. 10 Higher update block Area

The power for Higher update block is shown in below Fig.

the quiescent numbers in XPower are based on measurements of real designs with active functional elements reflecting real world design scenarios.

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		40
Vccint 1.80V:	19	33
Vcco33 3.30V:	2	7
Clocks:	3	6

Fig. 11 Higher update block Power

The Area for Lower update block is shown in below Fig.

```
Mapper Version : spartan2e -- $Revision: 1.34 $
Mapped Date    : Thu May 22 18:55:51 2014

Design Summary
--------------
Number of errors:      0
Number of warnings:   14
Logic Utilization:
  Number of Slice Latches:          20 out of 13,824    1%
  Number of 4 input LUTs:          158 out of 13,824    1%
Logic Distribution:
    Number of occupied Slices:                        95 out of  6,912    1%
    Number of Slices containing only related logic:   95 out of     95  100%
    Number of Slices containing unrelated logic:       0 out of     95    0%
        *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:         174 out of 13,824    1%
      Number used as logic:                    158
      Number used as a route-thru:              16
   Number of bonded IOBs:           48 out of    325   14%
      IOB Latches:                               2
   Number of GCLKs:                 1 out of      4   25%
   Number of GCLKIOBs:              1 out of      4   25%

Total equivalent gate count for design:  1,866
Additional JTAG gate count for IOBs:  2,352
Peak Memory Usage:  136 MB

NOTES:
```

Fig.12 Lower update block Area

The power for Lower update block is shown in below Fig.

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		39
Vccint 1.80V:	18	32
Vcco33 3.30V:	2	7
Clocks:	3	5
Inputs:	0	1
Logic:	0	0
Outputs:		

Fig. 13 Lower update block Power

4.3. Simulation Results

The implementation is done in Mat lab & the simulation results of arithmetic coder is shown in below Fig.

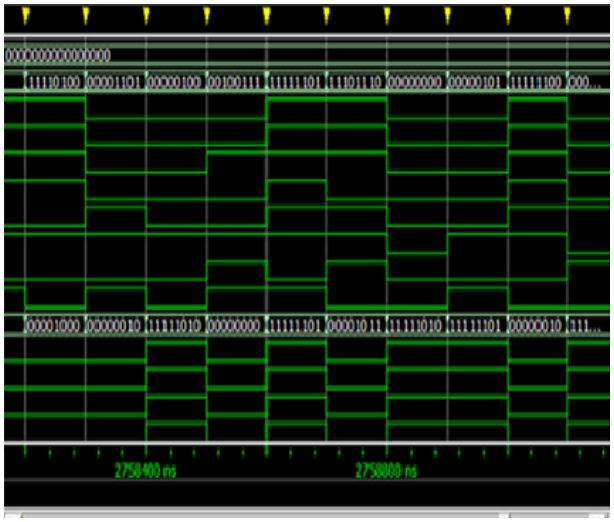

Fig. 14 Simulation Results of Arithmetic Coder

5. Conclusion

This paper presents implementation of arithmetic coder used in SPIHT. The hardware is realized on Spartan 6 FPGA kit. The design is realized by Xilinx ISE 13.2. A compression of 4:1 is achieved in this architecture so the memory occupied by Codeword is reduced by four times when compared to the input with bit-rate of 8bpp. As a result smaller storage space is needed to store the encoded bit-stream and it is easy to transmit encoded bit-

stream in lesser transmission bandwidth. For a pixel precision of 8 bits with a resolution of 512 x512, a throughput of coder is 800Mb/s. For improvement of throughput purpose SPIHT algorithm without lists can be implemented.

References

[1] C. Chrysafis & A. Ortega, "Line based, reduced memory, wavelet image compression",IEEE Trans. Image Process., Vol 9, No. 3, Sep.2000,pp.378–389.

[2] Kai liu,Eygeniy,Belyaey and Jie Guo,"VLSI architecture of arithmetic coder used in SPIHT",IEEE transactions on VLSI Systems, Vol 20,No.4,April 2012.

[3] Rehna.V.J , Shubhangi.S & Vasanthi.S,"Improving the performance of Wavelet based image compression using spiht algorithm", IRNet Transactions on E and E Engineering (ITEEE) Vol 1,Iss 2,2012.

[4] V. G. Oklobdzija, "An algorithmic and novel design of a leading zero detector circuit: Comparison with logic synthesis," IEEE Trans on VLSI systems, Vol. 2, No. 1, Mar. 1994, pp. 124–128.

[5] I.C.Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," Commun. ACM, vol. 30, no. 6, Jun. 1987, pp.520-540.

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

[7] K.SivaNagiReddy, V.Sidda Reddy, Dr.B.R.Vikram," Efficient Memory and Low Complexity Image Compression Using DWT with Modified SPIHT Encoder", International Journal of Scientific & Engineering Research, Vol 3, Issue 8, 2012.

[8] Bibhuprasad Mohanty, Abhishek Singh & Sudipta Mahapatra,"A high performance modified SPIHT for scalable image compression", International of Image processing (IJIP), Vol.5, 2011.

[9] Usha Bhanu.N and Dr.A.Chilambuchelvan,"A Detailed Survey on VLSI Architectures for Lifting based DWT for efficient hardware Implementation, VLSICS, Vol.3, No.2, April 2012.