Global Journal of Computer Science and Technology: F
Graphics & Vision
Volume 14 Issue 1 Version 1.0 Year 2014
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Design and FPGA Implementation of High Speed DWT-IDWT Architecture with Pipelined SPIHT Architecture for Image Compression

By T. Vijayakumar & S. Ramachandran

SJB Institute of Technology, India

Abstract- Image compression demands high speed architectures for transformation and encoding process. Medical image compression demands lossless compression schemes and faster architectures. A trade-off between speed and area decides the complexity of image compression algorithms. In this work, a high speed DWT architecture and pipelined SPIHT architecture is designed, modeled and implemented on FPGA platform. DWT computation is performed using matrix multiplication operation and is implemented on Virtex-5 FPGA that consumes less than 1% of the hardware resource. The SPIHT algorithm that is performed using pipelined architecture and hence achieves higher throughput and latency. The SPIHT algorithm operates at a frequency of 260 MHz and occupies area less than 15% of the resources. The architecture designed is suitable for high speed image compression applications.

Keywords: *SPIHT, DWT, pipelined architecture, FPGA, High speed, image compress ion.*

GJCST-F Classification: *I.4.0*

Strictly as per the compliance and regulations of:

Design and FPGA Implementation of High Speed DWT-IDWT Architecture with Pipelined SPIHT Architecture for Image Compression

T. Vijayakumar [α] & S. Ramachandran [σ]

Abstract- Image compression demands high speed architectures for transformation and encoding process. Medical image compression demands lossless compression schemes and faster architectures. A trade-off between speed and area decides the complexity of image compression algorithms. In this work, a high speed DWT architecture and pipelined SPIHT architecture is designed, modeled and implemented on FPGA platform. DWT computation is performed using matrix multiplication operation and is implemented on Virtex-5 FPGA that consumes less than 1% of the hardware resource. The SPIHT algorithm that is performed using pipelined architecture and hence achieves higher throughput and latency. The SPIHT algorithm operates at a frequency of 260 MHz and occupies area less than 15% of the resources. The architecture designed is suitable for high speed image compression applications.

Keywords: SPIHT, DWT, pipelined architecture, FPGA, High speed, image compression.

I. Introduction

Technological growth of semiconductor industry has led to unprecedented demand for low power, high speed complex and reliable integrated circuits for medical, defence and consumer applications. Today's electronic equipment comes with user friendly interfaces such as keypads and graphical displays. As images convey more information to a user, many of the equipment today have image displays and interfaces. Image storage on these smaller, handled devices is a challenge as they occupy huge storage space; also image transmission requires higher bandwidth. Hence most of the signal processing technologies today has dedicated hardwares that act as co-processors to compress and decompress images. Discrete Wavelet Transforms have been widely used for image compression and decompression. Much architecture has been proposed for high-speed 2-D DWT computation. The architectures can be broadly classified as separable and non-separable architectures [1]. Vishwanath etal. [2] Have proposed a low-storage short-latency separable architecture in which the row-wise operations are performed by systolic filters and the column-wise operations by parallel filters. Liao et al. [3] Have introduced architecture in which each of the row- and column-wise filtering operations are decomposed using the so called lifting operations into a cascade of sub-filtering operations. Chakrabarti et al. [4] have proposed two non-separable architectures, one using parallel 2-D filters and the other an SIMD 2-D array. Cheng et al. [5] have proposed an architecture in which a number of parallel FIR filters with a polyphase structure are used to improve the processing speed at the expense of increased hardware. Hung et al. [6], in an effort to provide a reduced count of multipliers and to facilitate the processing of the boundary data, have proposed an architecture that is a pipeline of one stage of parallel multipliers and two stages of accumulators to perform the accumulation tasks of the filters in each of the two directions. Marino [7] has proposed a two-stage pipeline architecture in which the first stage performs the task of the first decomposition level and the second stage decomposes all the remaining levels and has aimed at providing a short computation time. These existing architectures have not exploited the computational parallelism inherent in the DWT operation to the extent possible in order to provide a high speed. The proposed architecture is modular and allows extension to any precision without much effect on the clock frequency. Simulation results have established that the proposed fast implementation scheme can produce high-quality reconstructed signals. The parallel DWT processor enhances throughput by producing two outputs in one clock cycle. The speed of the presented architecture does not depend on the number of filter taps and the number of bits per sample value. 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 [8-9]. In [9], 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 [8]. Furthermore, SPIHT has about 0.6 dB peak signal-to-noise-ratios (PSNR) gain over EZW [10] and is very close to EBCOT in many circumstances [9].

Author α: Research Scholar, Kuvempu University, Karnataka. India. e-mail: vijayakumar_sjbit@yahoo.com
Author σ: Prof., Dept. of E & CE, SJB Institute of Technology, Bangalore, India.

In this work a pipelined SPIHT algorithm is designed for high speed applications. Section 2 discusses image compression and DWT algorithm, section 3 discusses DWT based image compression, Section 4 discusses design of DWT architectures and Section 5 discusses SPIHT encoder, Section 6 is conclusion.

II. Background Theory

An image (from Latin imago) is an artifact, for example a two-dimensional picture that has a similar appearance to some subject—usually a physical object or a person. Images may be two-dimensional, such as a photograph, screen display and as well as a three-dimensional, such as a statue. They may be captured by optical devices—such as cameras, mirrors, lenses, telescopes, microscopes, etc. and natural objects and phenomena, such as the human eye or water surfaces. A still image is a single static image, as distinguished from a moving image. This phrase is used in photography, visual media and the computer industry to emphasize that one is not talking about movies or in very precise or pedantic technical writing such as a standard. Image file size—expressed as the number of bytes—increases with the number of pixels composing an image and the color depth of the pixels. The greater the number of rows and columns, the greater the image resolution, and larger the file. Also, each pixel of an image increases in size when its color depth increases—an 8-bit pixel (1 byte) stores 256 colors, a 24-bit pixel (3 bytes) stores 16 million colors, the latter known as true color. The need for image compression becomes apparent when the number of bits per image resulting from typical sampling and quantization schemes is computed. Considering the amount of storage for the 'Lena' digital image, the monochrome (grayscale) version of this image with a resolution 512x512x8 bits/pixel requires a total of 2,097,152 bits, or 786,432 bytes. Such image should be compressed for efficient storage or transmission. Image compression refers to algorithms which reduce image data redundancy in order to store the image efficiently. Uncompressed multimedia data demands considerable storage capacity and bandwidth. Digital images can be compressed due to two factors which are [11]:

1. Spatial redundancy
2. Perceptual redundancy.

Spatial redundancy refers to intra frame redundancy that when the image are taken in dark back ground or when the background will be same colour most of the data required to represent the image will be same and are redundant which can be ignored to save storage space. This can also be defined as the correlation between the adjacent pixel values.

Figure 1 : Perceptual redundancy in images [2]

A pixel in an image will be represented as 8 bits. If we observe the two figures shown in Figure 1 above, the first one is represented with 6 bits and seems to be good in quality where as the other which is represented with 4 bits looks blurred. Human eye will not be able to distinguish even when the image is represented with 6 bits which is short of some information; this is called perceptual redundancy and is taken as advantage for compressing the images. Compression is of two types: Lossless compression and Lossy compression. Lossless compression is one where the information can be recovered after decompressing such that no loss of data occurs. For example 212 214 220 222 216 212 212 214 is the data in one line we can represent this as +212 +2 +6 +2 -6 -4 0 +2 and so on. This method is called predictive encoding. There is another way of lossless compression where for example k is the letter that repeats frequently in an image data so instead of representing k we replace all k's with '.' Simple dot as data represented for '.' is much less than a alphabet 'k'. In this way frequent symbols are represented with symbols which require less data to represent thus reducing the space required to store the image. This technique is called statistical encoding. In both the cases data can be recovered and no loss occurs. Lossy compression is one where the data compressed cannot be recovered. For example a 1024x1024 resolution image is transmitted as 512x512 pixels it is a lossy compression. There are different forms of lossy compression of which we are interested in transform

encoding based lossy compression. Transform encoding involves mathematical transformation separating image information based on gradual spatial variation of brightness from information with faster variation of brightness at edges of the image. In the next step, the information on slower changes is transmitted lossless, but information on faster local changes is transmitted with lower accuracy by quantizing the data. This results in loss of data which cannot be recovered.

III. Wavelet based Image Compression

Wavelet coding has become very popular due to its robustness under transmission and decoding errors at higher compression rates avoiding blocking artifacts. Wavelet based compression is based on sub-band coding. Sub band coding involves splitting the frequency band of the image into sub bands and then to code each sub band using a coder and bit rate accurately matched to the statistics of the band. Simple DWT consists of a low pass filter and high pass filter which splits the image into low frequency and high frequency sub bands.

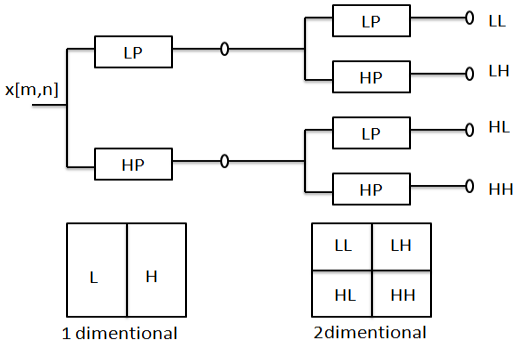

Figure 2 : 2-Dimensional DWT [2]

Figure 2 shows the two dimensional decomposition of image where the first one is row decomposition and the second stage is column decomposition.

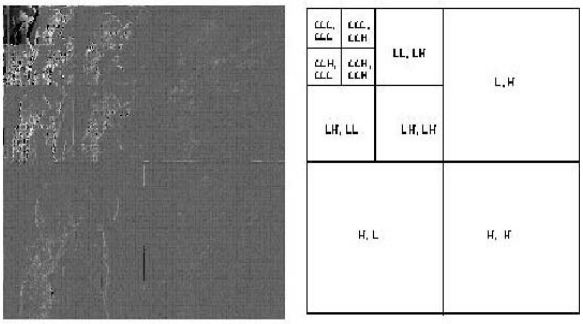

Figure 3 : Representation of decomposed image [2]

Decomposition is done till the image decomposes to 4x4 block. Figure 3 shows the decomposed image where the low frequencies are at the top left corner of the image and purely higher frequency at bottom right corner.

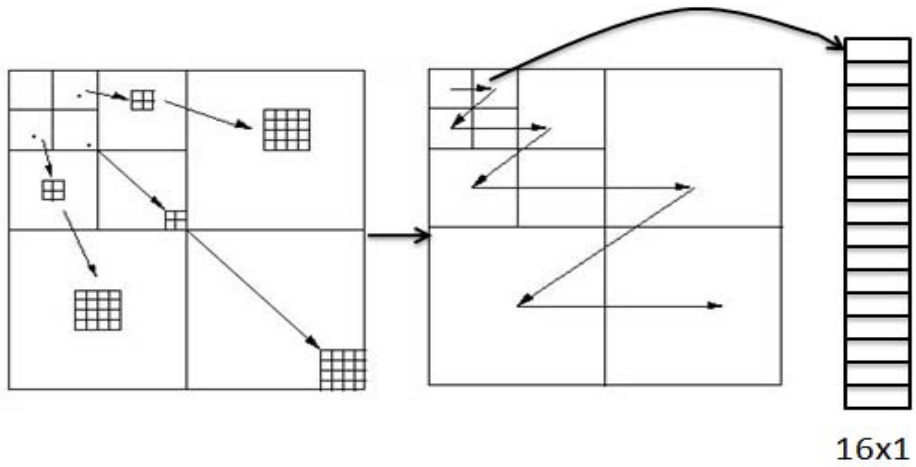

Figure 4 : Framing the decomposed blocks [2]

Once decomposed in to 4x4 blocks they are arranged in a 16x1 matrix in zigzag order as shown Figure 4, in this way all the blocks are arranged in the matrix finally it forms a 16xN matrix.

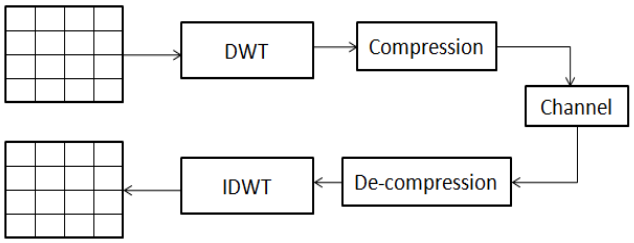

Figure 5 : Image compression block diagram

Finally after wavelet transformation image data is compressed and transmitted through channel and de-compressed at the receiver, after that IDWT will recover the image data that represents the image. In this work compression and de-compression algorithm have been developed as shown in Figure 5 using the existing wavelet transformation on FPGA hardware.

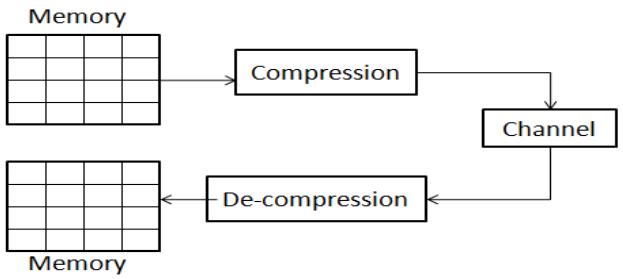

Figure 6 : Modified architecture

In order to demonstrate the working of the proposed hardware DWT values stored in a memory to compress and after de-compressing as shown in Figure 6.

IV. Design of dwt Architecture

In [12-15] DWT architectures have been discussed that use techniques like distributive arithmetic, lifting scheme and symmetric logic for hardware implementation to meet speed and power requirements. In [16-17] parallel processing algorithms are discussed for DWT implementation. In this work, high speed architectures for DWT and SPIHT encoding schemes are designed. Compressor involves converting a matrix of 16x64 elements into 4x64 elements by multiplying with fixed matrix of order4x64, this way memory required to store 16x64 elements is reduced and it is flexible to transmit 4x64 instead of 16x64 elements. Each column of 16x64 represents a 4x4 decomposed block and each element consists of 8 bits with sign-magnitude representation as shown in Figure 7.

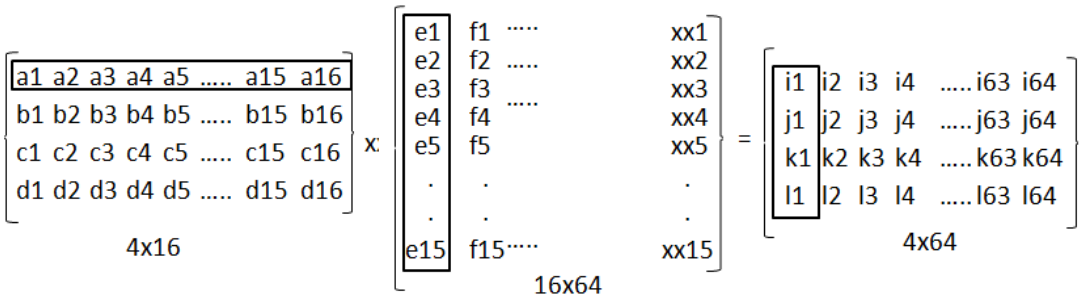

Figure 7 : Compressor concept

Compression basically done here is the multiplication of the input matrix with a fixed matrix to produce reduced number of elements for transmission and storage. Basic matrix multiplication involves multiplying and accumulating the row elements of one matrix with the other matrix. To perform this number of columns of first matrix should be equal to number of rows of the second matrix. One important consideration here as discussed in the previous step that 4x4 block is framed on to 16x1 matrix then input matrix is fed to the compressor as column by column. So that when the first column is received serially will be multiplied with 4 rows of the fixed matrix simultaneously as the input data is of real time nature, storing and processing input, would take much hardware hence we will proceed with 4-multipliers instead of one. So by the time when one complete column is received, it is processed and provides four outputs simultaneously. Output generation will require series of multiply and accumulate operations. If the input is in 2's complement one adder will be sufficient but as the input is signed format we need additional logic to decide whether addition or subtraction has to be performed. Since the last bit is sign bit we use only 7-bit multiplier instead of 8-bit thus balancing the hardware used for sign detection circuitry. Output will be stored in 8-bit, but after multiplication of two elements we get 15-bit including sign bit and if it is to be accumulated for 16 times which is the number of elements in the input column we get around 18 bits. So we will transmit only 8-MSB bits ignoring others, this is considered as quantization where certain information is lost. Another important aspect is storing the fixed matrix

elements in memory. If we store all the elements of matrix in a single memory, we can access only one element for one cycle. To perform 4 multiplications simultaneously and to access 4 row elements simultaneously, different row elements are stored in different memories of smaller size such that they can be accessed simultaneously. Another challenge occurs where output is available after every 16 cycles which is after receiving one complete column serially, 4 outputs will be available which cannot be sent serially in a single clock cycle. So we will transmit each output element for every 4 cycles because of a specific reason which we will discuss with de-compression section. After sending 4 elements for 16 cycles another set of outputs will be available to transmit. Since data is received serially we need to have a mechanism for synchronizing the data. An active high signal called 'vi' which is valid input is provided which is high if the input is valid, also a valid out signal is asserted when we transmit the data for each 4 cycles of the compressor. If this input is low data will not be received and no operation is performed.

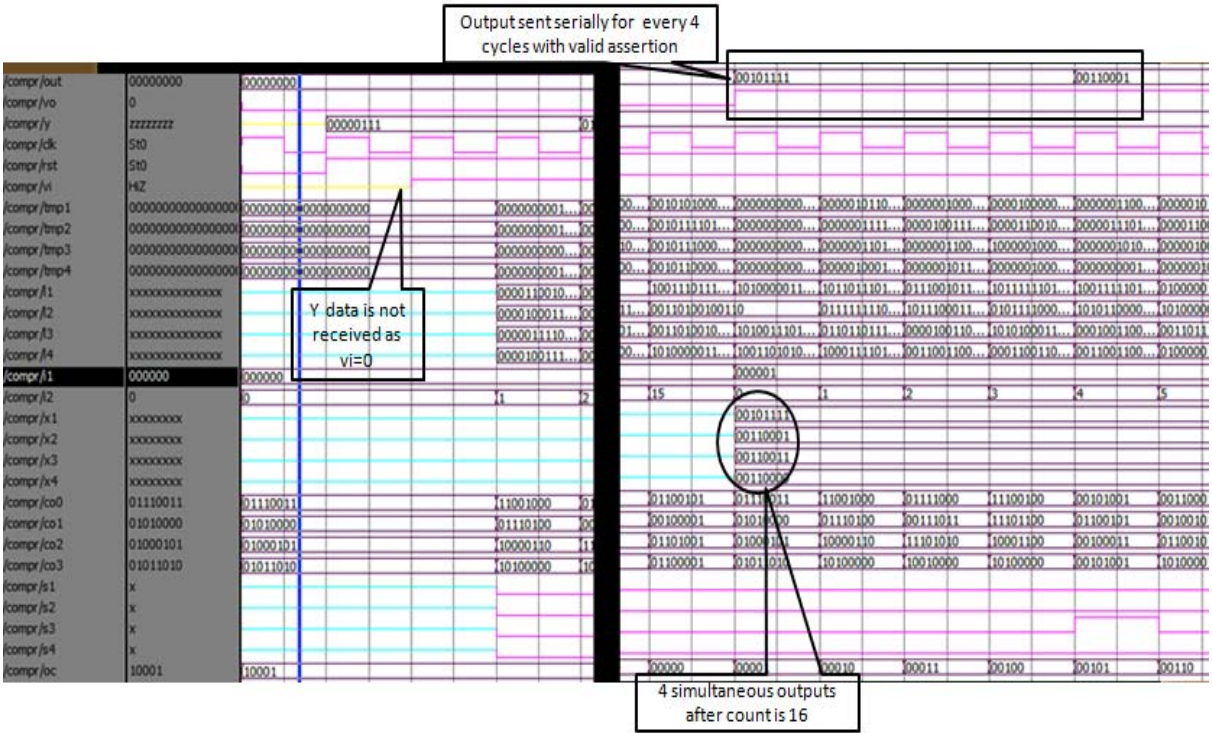

Figure 8 : Compression waveform

From the Figure 8 we can observe that when valid input is not asserted input y is not received and no operation is performed also, after 16 cycle 4 simultaneous outputs are available and are sent serially for each 4 cycles with 'vo' asserted which is marked in the waveform.

Device utilization for spartan3 and virtex2 FPGA are compared.Virtex2 has lesser utilization with 89 MHz frequency where as in spartan3 Maximum frequency is 83 MHz and Power consumed is 0.2W. It shows 4 multipliers one ROM and a total 4 adders and 8 subtractors were used, extra subtractors were required due to signed number representation.

Table 1 : Compressor device utilization for spartan3 xc3s200pq208-5

Number of Slices	300	1920	15%
Number of Slice Flip Flops	134	3840	3%
Number of 4 input LUTs	584	3840	15%
Number of bonded IOBs	20	173	11%
Number of MULT18X18s	4	12	33%

Table 2 : Compressor device utilization for virtex2 xc2v500fg256-5

Number of Slices	281	3072	9%
Number of Slice Flip Flops	134	6144	2%
Number of 4 input LUTs	549	6144	8%
Number of bonded IOBs	20	172	11%
Number of MULT18X18s	4	32	12%

SPARTAN3		Virtex2	
Macro Statistics		Macro Statistics	
# ROMs	: 1	# ROMs	: 1
16x32-bit ROM	: 1	16x32-bit ROM	: 1
# Multipliers	: 4	# Multipliers	: 4
7x7-bit multiplier	: 4	7x7-bit multiplier	: 4
# Adders/Subtractors	: 13	# Adders/Subtractors	: 13
18-bit adder carry out	: 4	18-bit adder carry out	: 4
19-bit subtractor	: 8	19-bit subtractor	: 8
5-bit adder	: 1	5-bit adder	: 1

a) De-Compressor

In de-compression compressed image is multiplied with the transpose of the fixed co-efficient matrix which was used for compression thus obtaining the actual 16x64 matrix is obtained.

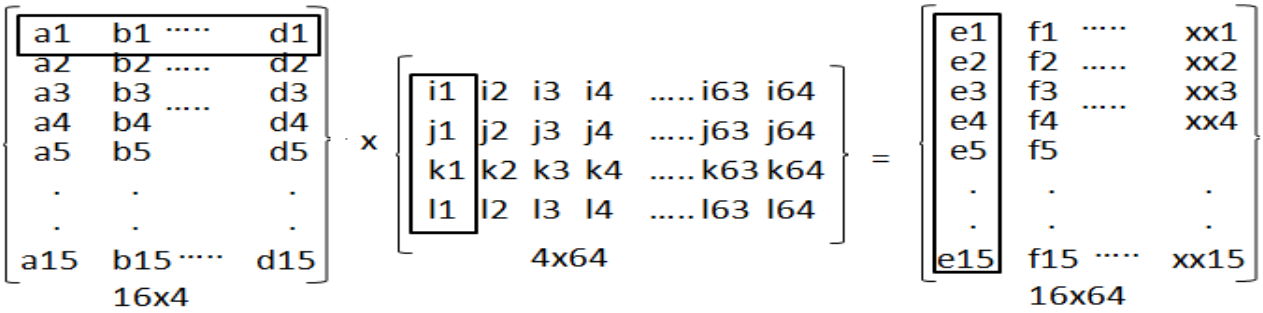

Figure 9 : De-compression

In De-compression we have 16 rows of fixed co-efficient as it is transposed so it infers that we need to use 16 multipliers to process real time input. This will consume more hardware so we will take advantage of the input property that input is available for every 4 cycles but not for every cycle which we intentionally designed for in the compressor section. Using this as advantage if we have the same 4 multipliers which multiplies an incoming element with 4 rows and before the next element comes that is after the next 4 cycles multiplication with all the sixteen rows will be completed thus with reduced hardware and same throughput is achieved by taking advantage of the input property. Using 16 multipliers has certain disadvantages which are when using 16 multipliers is input is not available for certain cycles for which no operation will be done causing reduced efficiency. Efficiency can be increased by operating de-compressor unit at 4 times faster than the previous block which is compressor, if it operates at high frequency 16 outputs will be available for each 4 cycles which have to be sent out serially in 4 cycles this imposes frequency constraints on the next computing block which is IDWT. So using 4 multipliers will have less hardware and high efficiency without latency and it is still capable of processing the real time compressed input. Output is again quantized to 8-bits to fit output register.

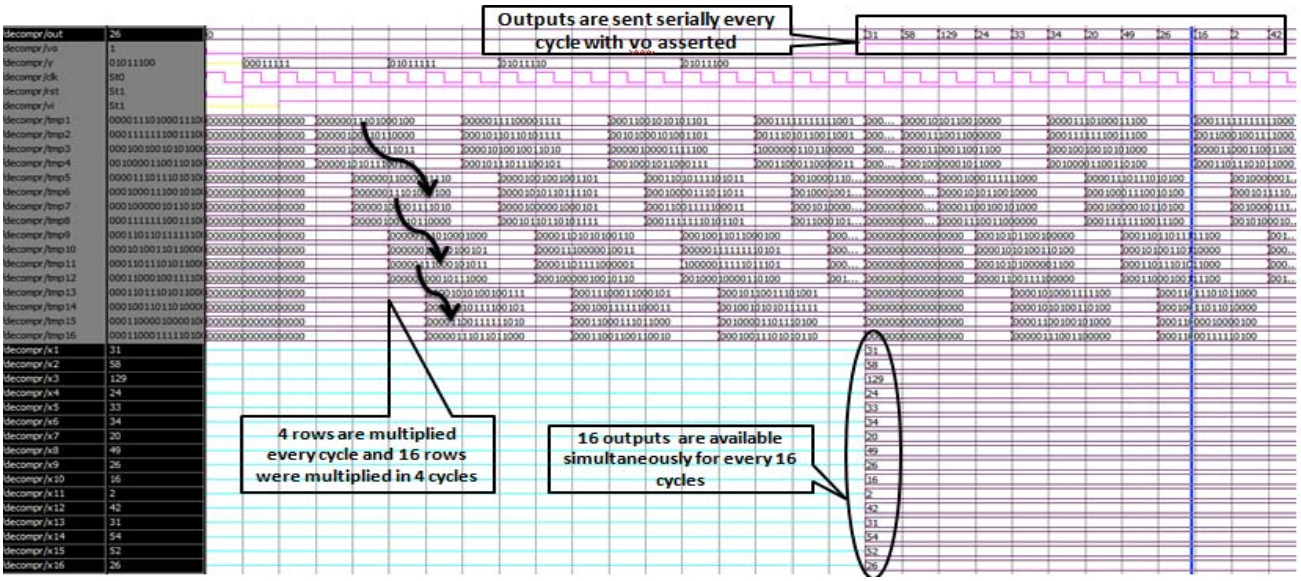

Figure 10 : De-compressor waveform

We can observe that 4 multipliers are used for four cycles to multiply 16 rows and after 4 cycles next input will be received and the process repeats. For every 16 cycles we will get 16 outputs simultaneously which will be stored in 16 registers and will be sent serially for every cycle and after 16 cycles we will obtain new set of outputs. Latency here is 16 but as the output is sent for every cycle throughput is 100%. This system can be called as a pipelined architecture.

Table 3 : De-compressor device utilization for virtex2 xc2v500fg256-5

Logic Utilization	Used	Available	Utilization
Number of Slices	1217	3072	39%
Number of Slice Flip Flops	424	6144	6%
Number of 4 input LUTs	2305	6144	37%
Number of bonded IOBs	20	172	11%

VIRTEX2

Macro Statistics
Multipliers : 4
7x7-bit multiplier : 4
Adders/Subtractors : 49
16-bit adder carry out : 16
17-bit subtractor : 32
2-bit adder : 1

SPARTAN3

Macro Statistics
Multipliers : 4
7x7-bit multiplier : 4
Adders/Subtractors : 49
16-bit adder carry out : 16
17-bit subtractor : 32
2-bit adder : 1

Device utilization for de-compressor is more than compressor as number of adders and subtractors as well as registers are 4 times more than the compressor because number of rows are 16 for fixed matrix and we have reduced the multipliers from 16 to 4.

b) Integration of Compressor and De-Compressor

Now we will integrate compressor and de-compressor along with memory to arrive at Figure 11. Here we have a memory containing DWT output which has a memory controller that controls the addresses to be read or written after which image is compressed and de-compressed and finally stored in a memory.

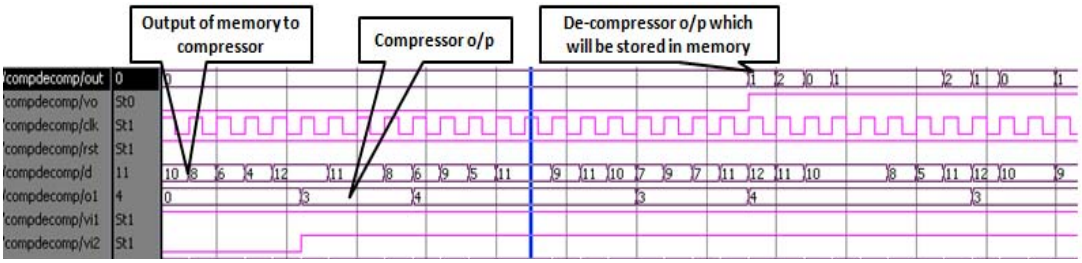

Figure 11 : Integrated waveform

D is the output of memory which is fed to compressor and o1 is the output of compressor fed to de-compressor and out is final de-compressed output which is again stored in the memory further to integrate with IDWT. Valid input and outputs used will provide synchronization between the multiple blocks and vo is asserted when data is available at out port.

Table 4 : Device utilization for Virtex-2 after integration

Logic Utilization	Used	Available	Utilization
Number of Slices	1500	3072	48%
Number of Slice Flip Flops	550	6144	8%
Number of 4 input LUTs	2853	6144	46%
Number of bonded IOBs	20	172	11%
Number of MULT18X18s	8	32	25%

Device utilization is combination of the utilization of compressor and de-compressor.

Table 5 : Device utilization for Virtex-5 after integration

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	838	69120	1%
Number of Slice LUTs	942	69120	1%
Number of fully used LUT-FF pairs	396	1384	28%
Number of bonded IOBs	50	640	7%
Number of BUFG/BUFGCTRLs	5	32	15%

V. SPIHT Architecture

DWT decomposes input image into sub bands of various frequency components. The decomposed sub bands contain the low frequency components and higher frequency components. The low sub band component is further decomposed using 2D DWT into another four sets of sub bands, the higher sub bands are retained as it is after one level of decomposition. The lowest sub band which is low frequency component is considered as the parent, all other sub bands are considered as the child. In order to compress the decomposed image, SPIHT algorithm is used for compression. SPIHT makes use of three lists – the List of Significant Pixels (LSP), List of Insignificant Pixels (LIP) and List of Insignificant Sets (LIS). These are coefficient location lists that contain their coordinates. After the initialization, the algorithm takes two stages for each level of threshold – the sorting pass (in which lists are organized) and the refinement pass (which does the actual progressive coding transmission). The result is in the form of a bit stream. SPIHT keeps three lists: LIP, LSP and LIS. LIP stores insignificant pixels, LSP stores significant pixels and LIS stores insignificant sets. At the beginning, LSP is empty, LIP keeps all coefficients in the lowest sub band and LIS keeps all tree roots which are at the lowest sub band. SPIHT starts coding by running two passes. The first pass is the sorting pass browses the LIP and moves all significant coefficients to LSP and outputs its sign. Then it browses LIS executing the significance information and following the partitioning sorting algorithms. The second pass is the refining pass browses the coefficients in LSP and outputs a single bit alone based on the current threshold. After the two passes are finished, the threshold is divided by 2 and the encoder executes the two passes again. This procedure is recursively applied until the number of output bits reaches the desired number. An input of image size 16 x 16 is encoded using SPIHT algorithm, the input image after DWT decomposition and different pass of SPIHT is shown in Figure 12. For a 16 x 16 input image each of 8 bits, the total number of input bits before encoding is 768, after sixth pass encoding the compressed bits are 277 at the receiver, 277 bits are decoded to obtain 768 bits. For higher compression, it is required to stop SPIHT with lower number of pass.

108	50	-40	23	18	16	-20	28	33	18	17	-31	-1	0	7	9
95	-20	-80	44	4	56	17	6	24	-15	14	10	4	6	5	0
30	76	15	47	8	52	-25	70	15	-9	15	12	0	8	2	1
45	-30	14	17	38	12	1	-5	7	4	6	1	4	3	2	-1
32	13	39	10	5	28	2	0	50	8	5	6	25	0	-48	0
4	70	20	1	8	18	23	9	15	7	0	1	9	18	6	1
7	2	44	3	7	-60	38	10	0	6	5	4	5	4	13	0
4	80	55	18	-30	15	-10	8	3	0	-2	21	-1	0	7	9
34	17	18	-12	15	0	1	7	7	2	0	1	8	4	-1	0
4	45	13	-1	0	13	4	5	0	0	3	2	16	25	8	1
-1	3	7	-11	2	0	1	2	4	5	6	0	7	5	3	2
0	8	14	-20	7	2	8	0	0	7	8	-1	1	0	2	1
1	3	0	0	4	7	0	0	2	0	7	8	-10	1	2	1
20	0	1	-1	3	2	0	1	1	9	0	0	0	0	0	-5
2	8	3	6	2	4	1	0	0	0	2	1	0	0	2	1
1	4	0	-2	1	3	-1	0	1	0	3	0	1	0	-1	0

Fifth pass: N=2 Threshold = 4 Number of bits = 283

110	50	-42	22	18	18	-22	30	34	18	18	-30	0	0	6	10
94	-22	-82	46	6	58	18	6	26	-14	14	10	6	6	6	0
30	78	14	46	10	54	-30	70	14	-10	14	14	0	10	0	0
46	-30	14	18	38	14	0	6	6	6	6	0	6	0	0	0
34	14	38	10	6	30	0	0	50	10	6	6	26	0	-50	0
6	70	22	0	10	18	22	10	14	6	0	0	10	18	6	0
6	0	46	0	6	-62	38	10	0	6	6	6	6	6	14	0
6	82	54	18	-30	14	-10	10	0	0	0	22	0	0	6	10
34	18	18	-14	14	0	0	6	6	6	0	0	10	6	0	0
6	46	-14	0	0	14	6	6	0	0	0	0	18	26	10	0
0	0	6	-10	0	0	0	0	6	6	6	0	6	6	0	0
0	10	14	-22	6	0	10	0	0	6	10	0	0	0	0	0
0	0	0	0	6	6	0	0	0	0	6	10	-10	0	0	0
22	0	0	0	0	0	0	0	0	10	0	0	0	0	0	6
0	10	0	6	0	6	0	0	0	0	0	0	0	0	0	0
0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Sixth pass: N=1 Threshold = 2 Number of bits = 277

109	51	-41	21	19	17	-21	29	33	19	17	-31	0	0	7	9
95	-21	-81	45	5	57	17	7	25	-15	15	11	5	7	5	0
31	77	15	47	9	53	-29	71	15	-9	15	13	0	9	3	0
45	-31	15	17	39	13	0	5	7	5	7	0	5	3	3	0
33	13	39	9	5	29	3	0	51	9	5	7	25	0	-49	0
5	71	21	0	9	19	23	9	15	7	0	0	9	19	7	0
7	3	45	3	7	-61	39	11	0	7	5	5	5	5	13	0
5	81	53	19	-31	15	-11	9	3	0	-3	21	0	0	7	9
35	17	19	-13	13	0	0	7	7	3	0	0	9	5	7	0
5	45	-13	0	0	13	5	5	0	0	3	3	17	25	9	0
0	3	7	-11	3	0	0	3	5	5	5	0	7	5	3	3
0	9	15	-21	7	3	9	0	0	7	9	0	0	0	3	0
0	3	0	0	5	7	0	0	3	0	7	9	-9	0	3	0
21	0	0	0	3	3	0	0	0	9	0	0	0	0	0	-5
3	9	3	7	3	5	0	0	0	0	3	0	0	0	3	0
0	5	0	-3	0	3	0	0	0	0	3	0	0	0	0	0

Total number of bits required for decoding =47+123+169+248+283+277= 1147

Figure 12 : SPIHT encoder for 16 x 16 image

For compression of higher image sizes, the input image of size NxN is decomposed to three levels using three 2D DWT. Each 2D DWT consist of two 1D DWT that is performed along row and columns of input matrix. First level decomposition give rise to LL, LH, HL and HH sub band of size N/2xN/2. The LL sub band is decomposed to four more sub bands of N/4xN/4 size and in the third level LLL sub band is decomposed into N/8xN/8 sub band. Thus after three level decomposition there exist four sub bands of size N/8xN/8, three sub bands of size N/4xN/4 and three sub bands of N/2xN/2. SPIHT encoder algorithm encodes the ten sub bands into bit stream based on the compression ratio expressed in terms of bits per pixel (bpp). Input image is represented using 8 bit per pixel, the compressed image can be represented using bpp less than 8, thus leading to compression. Figure 13 shows the top level architecture of SPIHT encoder, the input image is stored in an input register, for each pass the SPIHT encoder algorithm is realized using logic gates and the output is stored in an intermediate register. The intermediate memory is further processed by the second stage SPIHT encoder and multiple stages of SPIHT encoder is designed for processing the inout data in pipeline. The master clock is used for synchronization of pipeline stages.

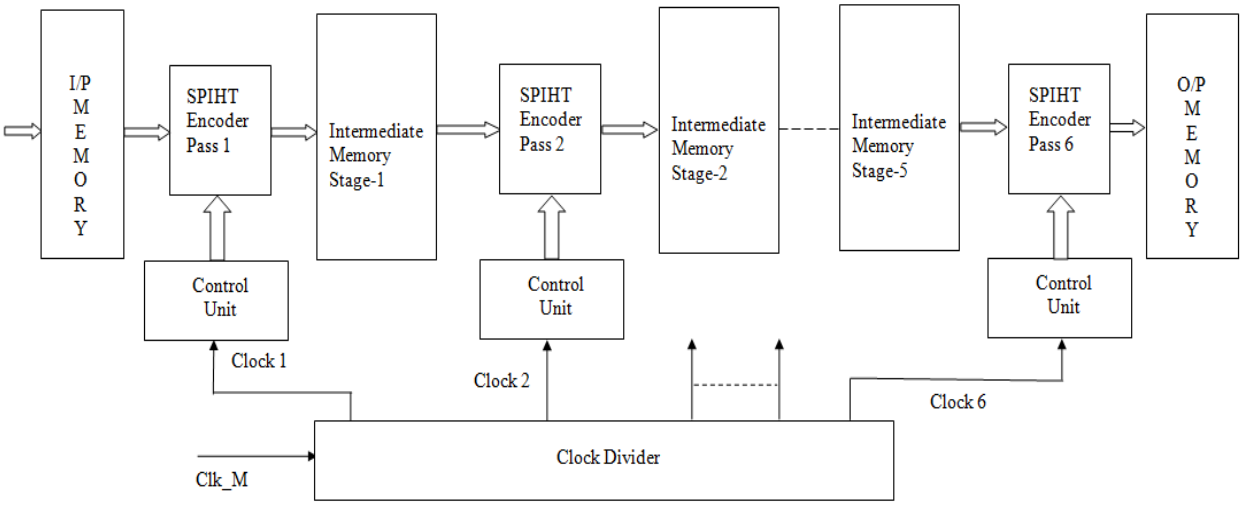

Figure 13 : Top level pipelined architecture for SPIHT encoder

Figure 14 shows the internal architecture of SPIHT encoder that consists of LIP memory for LL coefficients, LIS memory for higher sub bands. The LIP elements are compared for its significance and are transferred into LSP memory; similarly the LIS elements are partitioned and stored in sorted coefficients memory. Based on the LSP memory elements the bit stream is generated in the refinement pass. The pipelined architecture designed in this work achieves higher throughput and better latency. Independent memory modules used for every iteration increases area, however increases frequency of operation.

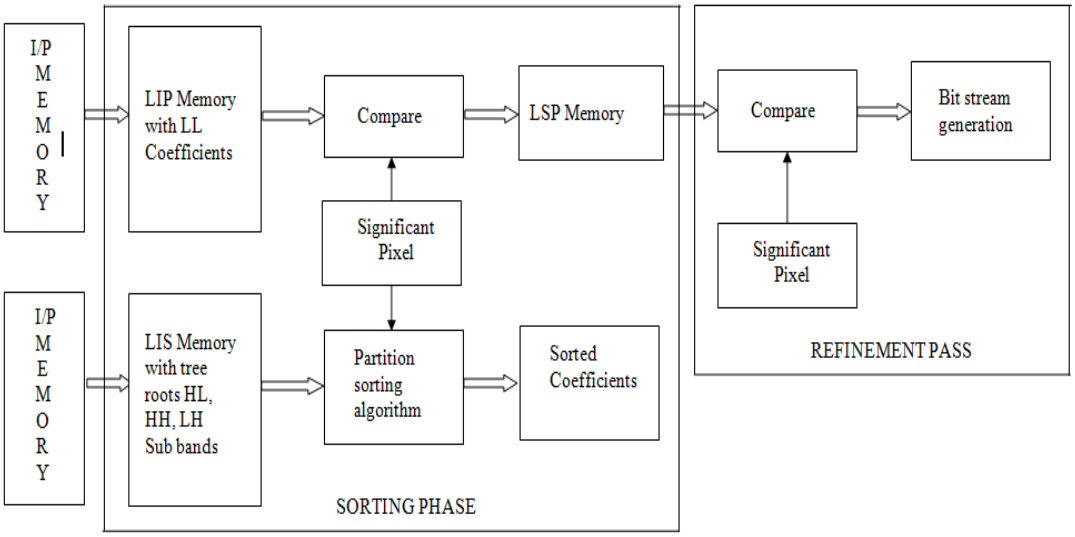

Figure 14 : Architecture of SPIHT encoder

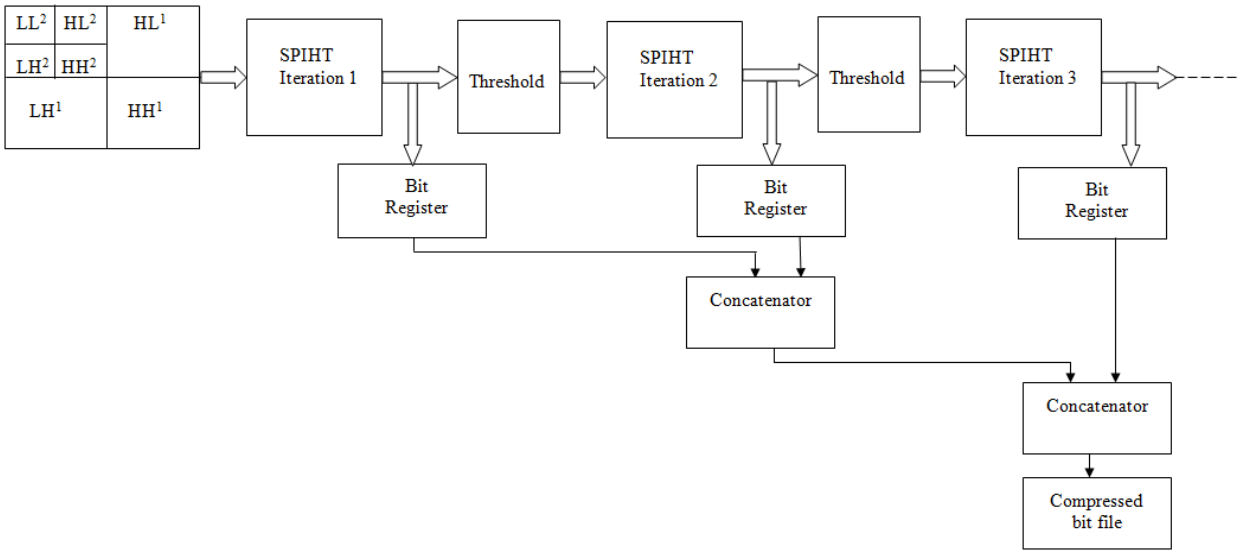

Figure 15 : Pipelined SPIHT encoder

The hierarchical sub bands obtained after two level 2-D DWT is stored in an input memory as shown in Figure 15. The SPIHT iteration 1 module generates the bit stream as discussed in Figure 14. The threshold module further scales the sub bands for generation of bit stream in the second iteration. The threshold levels are set according to the compression ratio required. In this work, variable threshold is set for LL and higher sub bands. The SPIHT encoder is repeatedly computes the bit stream until the sub bands in the input memory have elements greater than 10. The number 10 is fixed based on trial and error analysis carried out for achieving highest compression ratio of 120%. The final bit stream obtained after concatenation is regrouped and is transmitted over noisy channel. The SPIHT encoder is modeled using HDL and synthesized using Xilinx ISE targeting Virtex-5 FPGA. The synthesis results of first stage pipeline architecture for SPIHT encoder are shown in Figure 16.

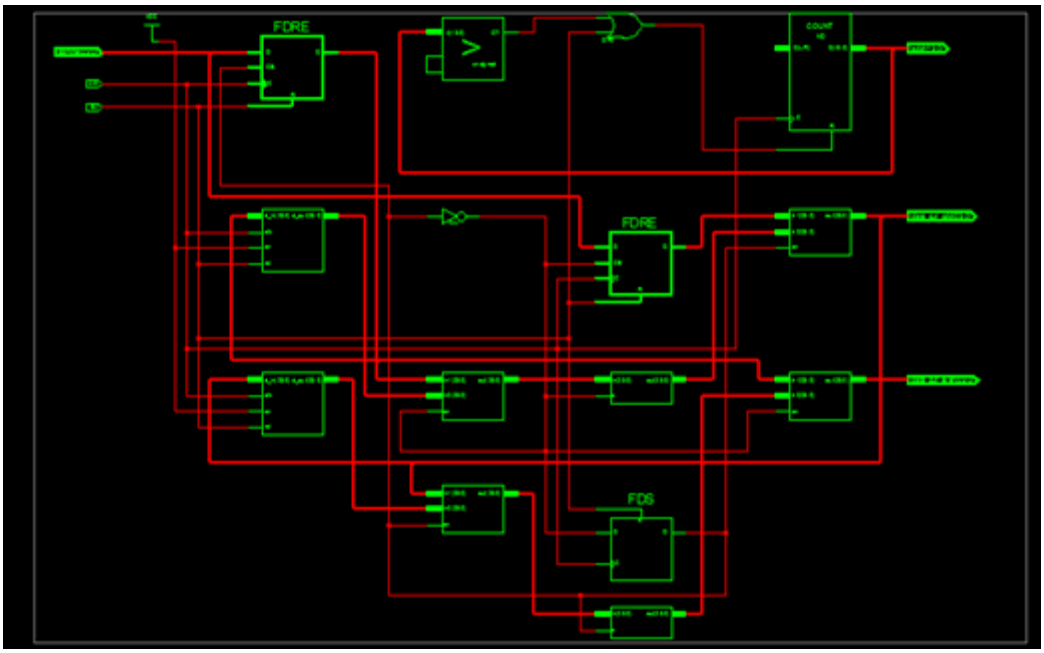

Figure 16 : First stage pipeline architecture for SPIHT encoder

Table 6 shows the synthesis results of SPIHT encoder on Virtex-5 FPGA. The HDL code is optimized for power and timing performances. From the simulation results and synthesis results obtained, it is demonstrated that the SPIHT algorithm developed functionally correct and is also optimized for high speed applications.

Table 6 : Synthesis results of SPIHT encoder

Parameters	Resource utilization
No. of slices	2389 out of 69120
No. of LUTs	3321 out of 69120
Max. Frequency	265 MHz
Power Dissipation	0.9mW

VI. Conclusion

Image compression technique with DWT is well understood and has been implemented on hardware. Entire architecture was split into individual blocks and developed for reduced hardware and higher throughput. De-compressor will take advantage of the input property that will be available for each 4 cycles and because of which 75% of the multipliers were reduced with 100% throughput and latency of 16. One important observation with XST toll is it will synthesize a block ROM when array is declared and used as memory where as if memory is defined in terms of look up tables XST will use CLB's. SPIHT encoder is designed using pipelined architecture and operates at maximum frequency of 265MHz. DWT combined with SPIHT occupy less than 5% of the resources on Virtex-5 FPGA.

References

1. Chengjun zhang, chunyan wang, and m. omair ahmad, A pipeline vlsi architecture for fast computation of the 2-d discrete wavelet transform, IEEE transactions on circuits and systems—i: regular papers, vol. 59, no. 8, pp: 1775-1785, august 2012.
2. M. Vishwanath, R. Owens, and M. J. Irwin, "VLSI architectures for the discrete wavelet transform," IEEE Trans. Circuits Syst. II, Analog. Digit. Signal Process. vol. 42, no. 5, pp. 305–316, May 1995.
3. H. Y. Liao, M. K. Mandal, and B. F. Cockburn, "Efficient architectures for 1-D and 2-D lifting-based wavelet transforms," IEEE Trans. Signal Process., vol. 52, no. 5, pp. 1315–1326, May 2004.
4. C. Chakrabarti and M. Vishwanath, "Efficient realizations of the discrete and continuous wavelet transforms: From single chip implementations to mapping on SIMD array computers," IEEE Trans. Signal Process, vol. 43, no. 3, pp. 759–771, Mar. 1995.
5. C. Cheng and K. K. Parhi, "High-speed VLSI implementation of 2-D discrete wavelet transform," IEEE Trans. Signal Process, vol. 56, no. 1, pp. 393–403, Jan. 2008.
6. K. C. Hung, Y. S. Hung, and Y. J. Huang, "A nonseparable VLSI architecture for two-dimensional discrete periodized wavelet transform," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 9, no. 5, pp. 565–576, Oct. 2001.
7. F. Marino, "Efficient high-speed low-power pipelined architecture for the direct 2-D discrete wavelet transform," IEEE Trans. Circuits Syst.II, Analog. Digit. Signal Process, vol. 47, no. 12, pp. 1476–1491, Dec. 2000.
8. 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.
9. D. Taubman, "High performance scalable image compression with EBCOT," IEEE Trans. Image Process., vol.9, no.7, pp.1158–1170, July 2000.
10. J.M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," IEEE Trans. Signal Process., vol.41, no.12, pp.3445–3462, Dec. 1993.
11. David S. Taubman, Michael W. Marcellin – JPEG 2000 – Image compression, fundamentals, standards and practice", Kluwer academic publishers, Second printing – 2002.
12. G. Knowles, "VLSI Architecture for the Discrete Wavelet Transform," Electronics Letters, vol.26, pp. 1184-1185, 1990.
13. A.S. Lewis and G. Knowles, "VLSI Architectures for 2-D Daubechies Wavelet Transform without Multipliers". Electron Letter, vol.27, pp. 171-173, Jan 1991.
14. K.K. Parhi and T. Nishitani "VSLSi Architecture for Discrete Wavelet Transform", IEEE Trans. VLSI Systems, vol. 1, pp. 191-202, June 1993.
15. Charilaos Christopoulos, Athanassios Skodras, and Touradj Ebrahimi - "THE JPEG2000 STILL IMAGE CODING SYSTEM - AN OVERVIEW", Published in IEEE Transactions on Consumer Electronics, Vol. 46, No. 4, pp. 1103-1127, November 2000.
16. J. Song and I. Park, "Pipelined discrete wavelet transform architecture scanning dual lines," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 56, no. 12, pp. 916–920, Dec. 2009.
17. C. Zhang, C.Wang, and M. O. Ahmad, "A VLSI architecture for a fast computation of the 2-D discrete wavelet transform," in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), May 2007, pp. 3980–3983.

Global Journals Inc. (US) Guidelines Handbook 2014

www.GlobalJournals.org