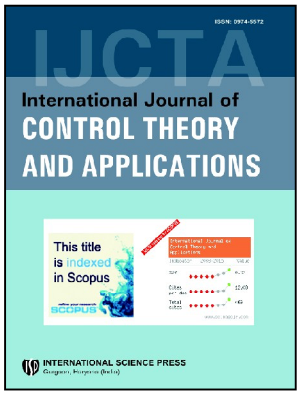


International Journal of Control Theory and Applications

ISSN : 0974-5572

© International Science Press

Volume 10 • Number 10 • 2017

Low Power VLSI Architecture for Image Compression Using Hyper analytic Wavelet Transform and Variable Length Encoding

[1]Nirmala S O and [2]R K Kamat

[1]Associate Professor, Department of Electronics & Communication Engineering, BIET under VTU, Belguam, E-mail: nirmalaso0474@gmail.com
[2]Professor, Department of Electronics, Shivaji University, Kolhapur

Abstract: The information handling capacity of computers has risen exponentially in the past decades. Hence the storage and transmission of the digital image component of Multimedia systems has become a major problem. Due to this the need for an efficient image compression technique is ever increasing. Even though a vast number of image compression techniques are present already, the users always opt for a technique with faster computing, low memory requirement and low power consumption. In this paper we have proposed the implementation of a low power, highly efficient image compression system in VLSI. Initially, input image pixels are converted to wavelet domain using Hyper Analytic Wavelet Transform (HWT) to determine the wavelet coefficients. Then, quantization block is developed and subsequently Variable Length Encoding (VLE) procedure is utilized to transform it into bit streams. The overall steps involved in the proposed VLSI architecture, is written as verilog code and the verification is done using Xilinx tools. Comparison of simulation results between the proposed with the traditional algorithm is also done with respect to power consumption, area and compression ratio. Here, compression rate will be computed using MATLAB tool. From the results, it can be inferred that our proposed technique have outperformed the existing techniques.

Keywords: Hyper Analytic Wavelet (HWT), Variable Length Encoding (VLE), Quantization, Low power, Image compression.

1. INTRODUCTION

A vital challenge thrown open in the course of the communication and accumulation of raw images is the indispensable need for massive amount of disk space. Thus, there is an ever-zooming requirement for a very powerful and healthy method for compression of such images. A superior compression method which is quicker, memory proficient and uncomplicated is certainly in a position to meet the needs of the user [25]. Generally, image compression refers to the compression of data on digital images. Principally, its goal is to diminish the redundancy of the image data in order to store or transmit the data in an effective manner as well as to provide a best image quality at a given bit rate (or compression rate). Two types of compression techniques often employed for image compression are: Lossless and Lossy [26]. The lossy compression that produces indiscernible differences can be called as visually lossless [19], [18]. In lossless image compression, the compression ratio obtained is extremely low and so, considerable resources cannot be saved by using such image compression. The image compression technique with compromising resultant image quality, without much notice of the viewer is the lossy image compression. The loss in the image quality increases with the percentage of the compression, hence results in saving the resources [19, 18]. In recent years, the wavelet theory and its application in image compression has progressed rapidly [19], [18], [6], [23], [17]. The field of wavelets is still sufficiently new and further progressions will continue to be reported in several areas. One of the most imperative processing components of image compression is wavelet transform [14].

Discrete Wavelet Transform (DWT), Quantization, and Entropy Encoding are the three main sequential steps followed during compression. After performing a preprocessing process, each component is analyzed individually by an appropriate discrete wavelet transform [21]. Due to the emergence of JPEG 2000 standard, a substantial attention has been paid to the development of an efficient DWT system architecture. FPGA (Field Programmable Gate Array) implementations can speed up the DWT by pipelining these operations. For real time signal processing, several DWT based VLSI architectures have been designed and implemented [24, 22, 5]. For 1-D DWT, the architectures are classified into three types: convolution based [7], lifting based [15], [19], [20], and B-spline based [16]. The convolution based method is used to implement two channel filter banks directly. The lifting based method exploits the relationship of low pass and high pass filters for saving multipliers and adders [28], [13] whereas, the third method i.e., B-spline can diminish the multipliers based on the B-spline factorization [16]. The B-spline based architectures offer less number of multipliers, while the lifting scheme fails to reduce the intricacy.

In wavelet image compression, the quantization, a lossy compression technique, is carried out after applying the wavelet, where a set of values is compressed to a single quantum value. When the innumerable discrete symbols in a given stream are diminished, the stream becomes more compressible. For e.g., it is feasible to diminish the file size of digital image by reducing the number of colors used to define such digital images. Certain applications perform DCT (Discrete Cosine Transform) data quantization in JPEG and DWT data quantization in JPEG 2000. After the quantization process, the quantized DWT coefficients are converted into sign-magnitude represented before entropy coding because of the intrinsic characteristics of the entropy encoding process. Normally, entropy coding can provide a much shorter image representation by means of short code words for probable images and longer code words for less probable images [12]. Entropy encoding, a lossless type of compression, is done on a certain image for making more competent storage. Normally, either 8-bits or 16-bits are necessary to store a pixel on a digital image. But, by means of efficient entropy encoding, a small number of bits are adequate to represent a pixel in an image and thus, this results in less memory usage to store or even transmit an image. Engineers have employed Shannon-Fano entropy, Huffman coding, Kolmogorov entropy, and arithmetic coding in various applications [27]. In this paper, we have proposed a wavelet based image compression algorithm. Here, the diminution of wavelet coefficients will be done such that to increase the compression rate. Then, quantization block will be developed and subsequently variable length encoding procedure will be utilized to transform it into bit streams in such a way that, the images will get compressed.

This paper is organized as follows: in Section 2, the existing works related to our proposed work is discussed briefly. The methodology for our research is clearly discussed in Section 3. The proposed architecture low power VLSI architecture for image compression using hyperanalytic wavelet transform and variable length encoding is described in detail in Section 4. Section 5 describes the experimental results and its comparison with the similar existing comparison. A conclusion is given in Section 6.

2. RELATED WORK

Literature is a significant treasure house of various VLSI architectures for image compression. At this juncture, we recount certain architecture offered in the literature. Vijaya Prakash. A.M and K.S. Gurumurthy [1] vigilantly investigated the execution of Low Power VLSI architecture for image compression, which employs Variable

Length Coding technique to compress JPEG signals. Their innovative architecture comprised three optimized blocks, such as Zigzag scanning, Run-length coding and Huffman coding. In their architecture, Zigzag scanner employed two RAM memories in parallel to speed up the scanning. The Run-length coder in their architecture, counts the number of intermediate zeros in between the successive non-zero DCT coefficients dissimilar to the conventional run-length coder which counts the recurring string of coefficients to compress data. The intricacy of the Huffman coder was decreased by employing a lookup table created by organizing the {run, value} combinations in the order of reducing probabilities with related variable length codes. The innovative hardware architecture for image compression was integrated by means of RTL complier and it was mapped by means of 90nm standard cells. Power consumptions of variable length encoder and decoder are restricted to 0.798mW and 0.884mW with least area.

Devangkumar Umakant Shah and Chandresh H. Vithlani, [2] launched a discrete wavelet transform based VLSI-oriented lossy image compression technique. It was extensively employed as the hub of digital image compression. Now, the functional simulation of each module was offered and the execution of each module was broadly assessed with gate needed, clock cycles desired, power, processing rate, and processing period. The innovative compression technique was assessed and contrasted with the modern approaches in relation to processor area and power. Relative outcomes exhibited that the innovative approach ushered in superb accomplishment in power-efficiency analogous to 0.328mW/chip vis-a-vis the erstwhile techniques. The compression rate was increased by bringing in RW block which puts roadblocks against certain coefficients gathered from the high pass filter to zero. Now, Distributed Arithmetic (DA) technique was performed to estimate the wavelet coefficients, in order that the number of arithmetic operation can be cut down significantly.

A configurable single precision floating-point (SPFP) DHT processor was excellently offered by Wang Xu *et.al* [4] for stepping up the estimation of filter in Katsevich formula. The configurable processor consisted of memory based architecture with one streamlined butterfly processing engine (PE) and it supported variable point dimensions from 8 to 1024. The DHT processor was effectively managed by the address generator. In accordance with the point dimension, the address generator generated one memory address pair per clock cycle to maintain the processor accessing memories consecutively. The DHT was determined without difficulty by means of intricate multiplications in the frequency domain. Two fast Fourier transforms (FFT) were highly indispensable for the overall task. The radix-2 FFT algorithm with decimation-in-frequency (DIF) decomposition was employed in the design to build an effective signal flow graph (SFG) for DHT estimation. Arithmetic computations, in the final FFT iteration, complicated multiplications and the initial IFFT iteration were substituted with conjugation and swapping operations, so that the two iterations were stored in the DHT SFG. Data were loaded and unloaded concurrently after one frame data computation came to a finish. The symmetric property of twiddle factors was employ to cut down half dimension of the read-only memory (ROM). Truncation was made use of in the design to decrease data path width. In comparison and contrast with preceding works, the performance assessment vividly demonstrated the fact that their DHT processor had the least clock latency.

Bhuyan M.S *et.al* [5] brilliantly brought out the hardware design flow of lifting based 2-D Forward Discrete Wavelet Transform (FDWT) processor for JPEG 2000. With an eye on creating superior quality image of JPEG 2000 codec, an efficient 2-D FDWT algorithm was executed on input image file to derive the decomposed image coefficients. The Lifting Scheme cut down the number of processes accomplishment stages to around 50% of those necessary for a conservative convolution method. Moreover, the Lifting Scheme was open to “in-place” calculation, in order that the FDWT could be executed in low memory systems. At first, the lifting based 2-D FDWT algorithm was expanded by means of Matlab. The adapted codes were then decoded into behavioral stage of FDWT algorithm in VHDL.

Dhulap S.S. and Nalbalwar S.L in [14] deftly devised a wavelet-based compression scheme that was competent to perform in lossy and lossless modes. At the outset, they spelt out integer wavelet transform (IWT) and integer wavelet packet transform (IWPT) as an application of lifting scheme (LS). After assessing and executing outcomes for IWT and IWPT, a parallel technique blending DPCM and IWPT was performed by means of Huffman coding for grayscale images. Thereafter they proceeded to execute the identical one for color images by means of Shannon's source coding method. They estimated the degree of compression by the compression ratio (CR) and compression factor (CF). Comparison and contrast with IWT and IWPT the DPCM-IWPT illustrated superb efficiency in image compression. Pujar J.H. and Kadlaskar L.M in [25] proficiently put forward the Lossless technique of image compression and decompression by means of an easy coding method termed Huffman coding, which was endowed with the quality of being trouble-free in execution and employing minimum memory. A software algorithm was evolved and executed to compress and decompress the specified image by means of Huffman coding method in a MATLAB platform.

Kai Liu *et al*., [29] characteristically offered superior speed architecture of AC employed in SPIHT without lists algorithm for enhancement of throughput. The intricate context model employed by software was moderated by designing an easy context model, which made use of the brother nodes' states in the coding zero tree of SPIHT to build context symbols for the arithmetic coding. Now, high-throughput memory-efficient arithmetic coder architecture for the set partitioning in hierarchical trees (SPIHT) image compression was launched founded on an easy context model. An out-of-order execution mechanism for various kinds of situations was envisaged which was able to apportion the context symbol to the idle arithmetic coding core with a diverse order than that of the input. The architecture gained from several optimizations executed at diverse phases of arithmetic coding from higher algorithm abstraction to lower circuits executions. The straightforward context model led to a standard access pattern while reading the wavelet transform coefficients which was very appropriate to the hardware execution, though followed by a minimal execution loss. Anyhow, in the case of hardware execution, the intricacy of computation restricted s AC in the field of high speed real-time coding. It was an elevated parallelism and computation mechanism which accelerated the pace of context processing. From the simulation outcomes, it was found that their AC architecture was able to satisfy several superior speed image compression needs. Thereafter, high speed computation units were engaged as catalysts.

Sunil W. Bhise *et al,* [30] have systematically brought to light the architecture and Verilog design of a Two Dimensional Discrete Cosine Transform (2DDCT) with Quantization and zigzag array. The 2D- DCT computation was performed by means of the 2D- DCT separability characteristic. In the same way, the entire architecture was segmented into two 1D-DCT computations by means of a transpose buffer. The design took 2115 slices, 1908 slice flip flops and 3134 LUTs including the control block which is appropriate for low cost FPGA like Xilinx XCS500E. 2D-DCT integrated with quantization and zigzag buffer was launched by means of Verilog system and tested with genuine gray scale image.

3. PROPOSED METHODOLOGY

The principal objective of this work is to design and expand VLSI architecture for Lossy Image Compression Using hyper analytic wavelet and Variable length encoding. This document is triggered by the work published in [1], which explores the execution of Low Power VLSI architecture for image compression, by means of DCT and Variable Length Encoding technique to compress JPEG signals. While, in [2] DA-based architecture for wavelet was employed for alteration and consequently, quantization and encoding procedure were executed. Boosted by the works offered in [1] and [2], a strenuous effort has been taken to widen VLSI architecture for hyper analytic wavelet (HWT) [3]. The HWT is designed by employing the architecture for Discrete Hilbert Transform (DHT) envisaged in [4]

The most important contributions of our work are:

- Low power VLSI architecture for hyper analytic wavelet by envisaging a configurable floating-point DHT and 2-D DWT architecture.
- Expanding a behavioral Variable length encoding model for enhanced efficiency.

Most of the existing works adopts DWT for computing wavelet coefficient and the DWT coefficients lacks shift sensitivity, Poor directionality and phase information and this leads to the increase in error. The HWT used in our proposed technique overcomes all these disadvantages and hence our technique is more efficient than other compression techniques. Also our architecture includes an efficient fused floating point add-subtract unit for contributing the low area also most of the encoding techniques used adopts LUT for their implementation thereby contributing to the power consumption.

4. PROPOSED METHOD

Now, let us discuss the details of the proposed image compression method by means of Hyperanalytic Wavelet Transform and Variable Length Encoding. At the outset, the image is split into 8×8 blocks of pixels and the HWT is performed on each block and each pixel in the image is transformed into coefficients. At this instant, these coefficients can be compressed further effortlessly as the data is statistically integrated in approximately a small number of coefficients. At last, the Quantized image is furnished to Variable Length Encoder in order that the compression task is completed. The entire block diagram of our proposed image compression method is exhibited in figure 1.

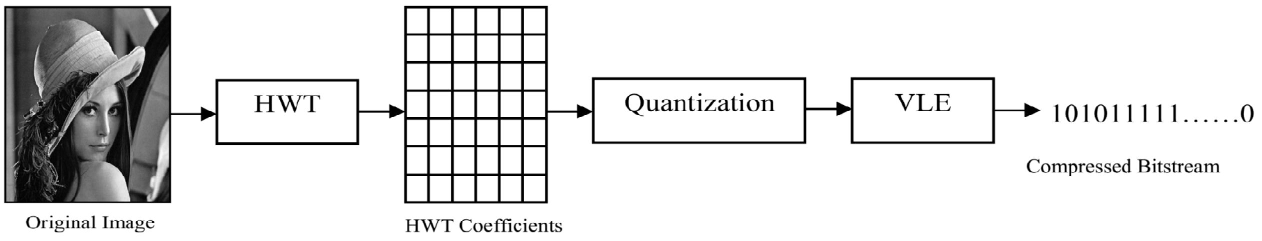


Figure 1: Overall Block Diagram of the Proposed Image Compression System

HyperAnalytic Wavelet Transform

The input image is first broken into 8×8 blocks of pixels and then each pixel is fed as input for computing HWT coefficients. Here the Hypercomplex representation of Hyperanalytic mother wavelet associated to the real wavelet $\varphi(x, y)$ is defined as,

$$\varphi_a(x,y) = \varphi(x,y) + lH_x\{\varphi(x,y)\} + mH_y\{\varphi(x,y)\} + nH_x\{H_y\{\varphi(x,y)\}\} \quad (1)$$

Where, $l^2 = m^2 = n^2 = lmn = -1, lm = -ml = n, mn = -nm = l, nl = -ln = m$

The Hyperanalytic Wavelet Transform (HWT) of an image *f(x, y)* is

$$HWT\{f(x,y)\} = \langle f(x,y), \varphi(x,y)\rangle + \langle f(x,y), lH_x\{\varphi(x,y)\}\rangle$$

$$+\langle f(x,y), mH_y\varphi(x,y)\rangle + \langle f(x,y), nH_x\{H_y\{\varphi(x,y)\}\}\rangle$$

$$= \langle f(x,y), \varphi(x,y)\rangle + l\langle f(x,y), H_x\{\varphi(x,y)\}\rangle$$

$$+m\langle f(x,y), H_y\varphi(x,y)\rangle + n\langle f(x,y), H_x\{H_y\{\varphi(x,y)\}\}\rangle$$

$$HWT\{f(x,y)\} = DWT\{f(x,y)\} + lDWT\{H_x\{f(x,y)\}\}$$

$$+mDWT\{H_y\{f(x,y)\}\} + nDWT\{H_x\{H_y\{f(x,y)\}\}\}$$

$$= \langle f_a(x,y), \varphi(x,y)\rangle = DWT\{f_a(x,y)\} \quad (2)$$

From eqn (2) it is clear that the HWT of the image f (x, y) can be estimated by adopting 2D DWT of its associated hyper complex image. Thus, the HWT architecture can be executed by means of four sections, each performing 2-D DWT. The first section is performed on the input image *f(x, y)*. The second and the third process are executed on 1D Hilbert transforms estimated across the rows (*Hx*) or columns (*Hy*) of the input image block. The final section is executed on the outcome gathered after the estimation of the two 1D Hilbert transforms of the input image. The HWT performance is presented in Figure 2.

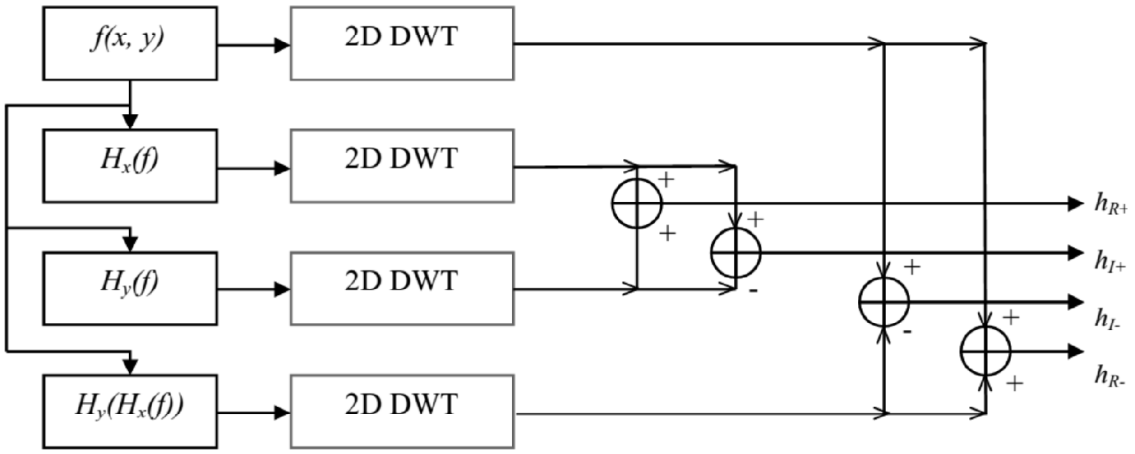


Figure 2: Architecture of HWT

Hyperanalytic Wavelet Transform (HWT) endowed with the merit of an easier execution is launched here. It is quasi shift-invariant, and possess a superb directional selectivity, and a decreased degree of redundancy in relation to the Discrete Wavelet Transform. From figure 2,

$$h_{R+} = detailcoefficients(DWT\{f(x,y) - DWT\{H_x\{H_y\{f(x,y)\}\}\})$$

$$h_{R-} = detailcoefficients(DWT\{f(x,y) + DWT\{H_x\{H_y\{f(x,y)\}\}\})$$

$$h_{I+} = detailcoefficients(DWT\{H_x\{f(x,y)\}\} + DWT\{H_y\{f(x,y)\}\})$$

$$h_{I-} = detailcoefficients(DWT\{H_x\{f(x,y)\}\} - DWT\{H_y\{f(x,y)\}\}) \quad (3)$$

$$h_R = h_{R+} + jh_{R-} \ \ and \ \ h_I = h_{I+} + jh_{I-} \quad (4)$$

The HWT coefficients for each 8×8 block are thus computed using our efficient HWT architecture. The implementation of floating point 1D-DHT and 2-D DWT is explained below.

Discrete Hilbert Transform Processor (*H (f*))

The most extensively applied technique for estimating the discrete HT is by means of the FFT. For hardware performance, architectures of discrete HT processor based on FFT algorithms are usually segregated and clustered into pipelined and memory based architecture styles. The discrete HT can be estimated by means of FFT as shown below:

$$\hat{x}(u) = IFFT(-j\,\mathrm{sgn}(k)X(k)) \quad (5)$$

Where,

$$X(k) = FFT\ (x(u))$$

$$-j\,\mathrm{sgn}(k) = \begin{cases} -j & k \in [1, N/2 - 1] \\ 0 & k = 0, N/2 \\ +j & k \in [N/2+1, N-1] \end{cases}$$

This method converts the input sequence to the frequency domain, then estimates the Hilbert transform in the frequency domain and at last carries out an IFFT function to derive the desired Hilbert-transformed sequence. The discrete HT architecture comprises seven phases, which are estimated by P0, P1, P2, P3 and P4. P0 is a module without twiddle factor multiplication, and is a sub module of P1, P2 and P3. These three kinds of PEs are modules that possess twiddle factor multiplications, and are segregated into two interior components. One component is fused floating point addition in which additions and subtractions are represented by a rectangle with a string “P0” within. The second component is twiddle factor multipliers or Floating Point multipliers represented by rectangles with letter “m” followed by an interior number. The star symbol in the figure refers to a conjugating process which is simple to execute by taking the 2’s complement of the imaginary component of a complicated value. The divided-by-16 module can be replaced with a shifter. The block diagram of 16-point Hilbert transform processor is illustrated in figure 3.

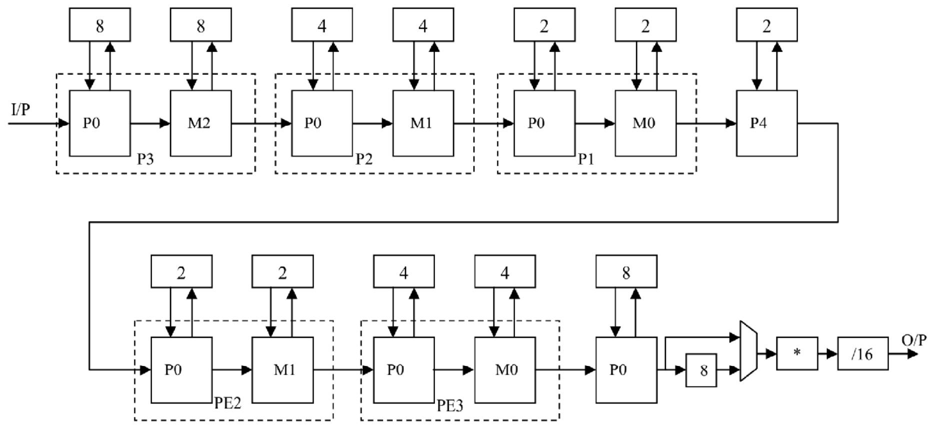


Figure 3: Block diagram of the 16-point Hilbert transform processor

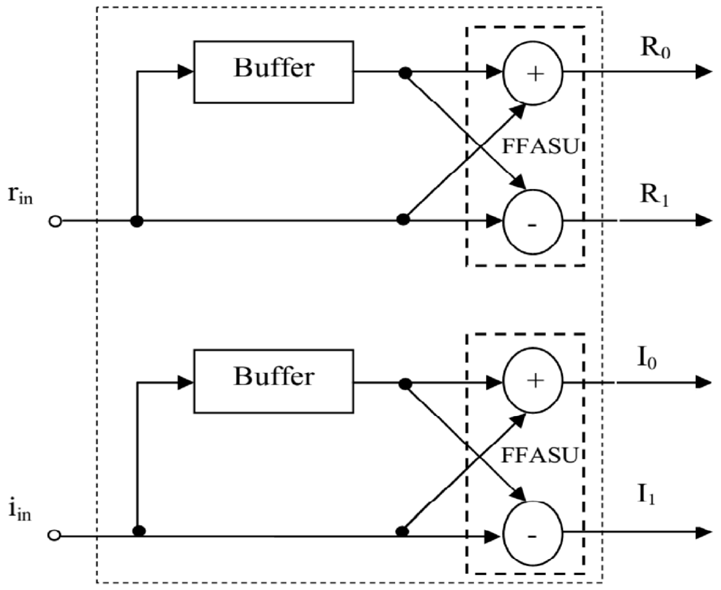


Figure 4: The Architecture of P0

Fig Architecture of P0 shown in figure 4 is employed to carry out the complex additions of the butterfly by means of the FFASU (Fused Floating point Add-Subtract Unit), and functions as the sub-modules of other PEs. (r_{in}, i_{in}) is the complex input data, (R_{out}, I_{out}) is the complex output data. (R_0, I_0) and (R_1, I_1) are one pair complex output data in the figure. Complex input data will be stored to DL (Delay-Line) buffers firstly till DL buffers are filled in full. By this method, the input data is disintegrated into two parallel data stream flowing forward, with identical length and accurate "distance" between data components getting admitted to the adders by appropriate delays.

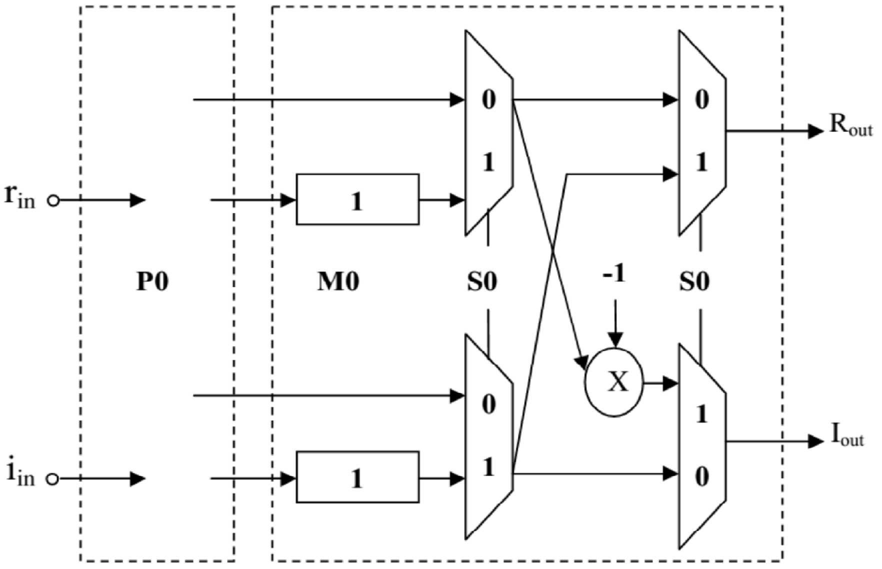


Figure 5: The Architecture of P1

As for the P1 stage exhibited in Fig.5, "M0" is a sub-module to calculate multiplication by *–j*. Data (R_0, I_0) from P0 is sent to "M0" at first. Simultaneously, data (R_1, I_1) is saved in buffers in "M0". The length of buffers in "M0" is 1. If signal "S0"is 0, data from P0 will be sent to (Rout, Iout) straight. If signal "S0"is 1, data from DL buffers will be multiplied by *–j*, and then sent to (Rout, Iout). DL buffers in Fig.5 are employed to integrate two parallel complex data (R_0, I_0) and (R_1, I_1) into one consecutive serial data

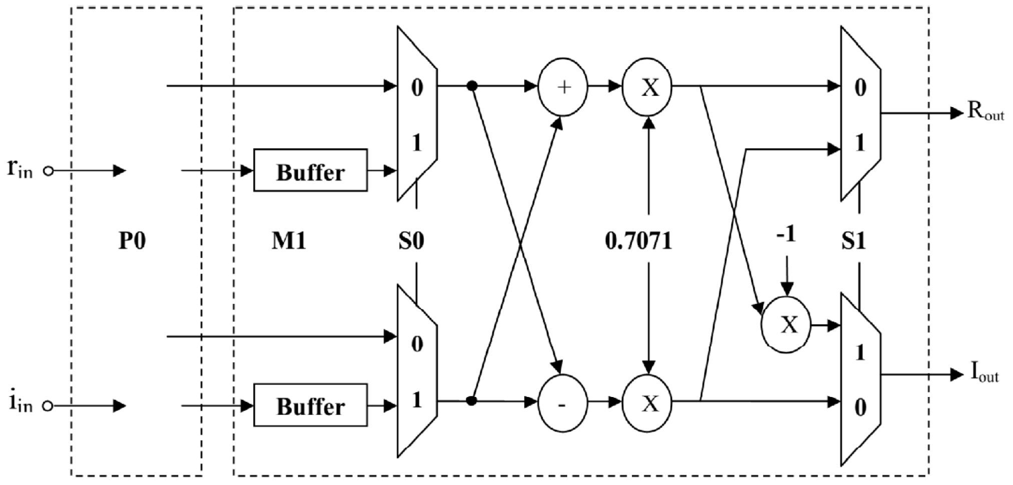


Figure 6: The Architecture of P2

The pipelined structure of the P2 is illustrated in figure.6. “M1” is a sub-module employed to estimate the twiddle factor multiplications with $W_N^{N/8}, W_N^{3N/8}$ or *-j*. $W_N^{3N/8} = -jW_N^{N/8}$, the multiplication by $W_N^{N/8}$ and then by *–j* can be substituted for the multiplication with $W_N^{3N/8}$. Hence, the structure of P2 utilizes this type of cascaded computation and certain multiplexers to derive all the essential estimations in the P2 stage. This technique stores one complex multiplier to create a low-cost hardware for computing $W_N^{3N/8}$. In the figure, signal “S0” is employed to choose data from (R_0, I_0) or DL buffers. Signal “S0” is also employed to facilitate or hinder adders and multipliers in “M1”.

Table 1
Twiddle factor selection

S0	*S1*	*Twiddle factors*
0	0	1
0	1	*-j*
1	0	$W_N^{N/8}$
1	1	$W_N^{3N/8}$

Signals “S0” and “S1” function jointly to manage the twiddle factors to be multiplied as exhibited in Table 1. If “S1” and “S2” are both zeroes, indicating the twiddle factor is 1, no multiplication needs to be performed. If “S1”is zero and “S2”is one, then “M1”performs the complex multiplications by *-j*. If “S1” is one and “S2” is zero, then “M1” performs the floating point multiplications by $W_N^{N/8}$. If “S1” and “S2” are both one, then “M1” performs the multiplications by twiddle factor $W_N^{3N/8}$.

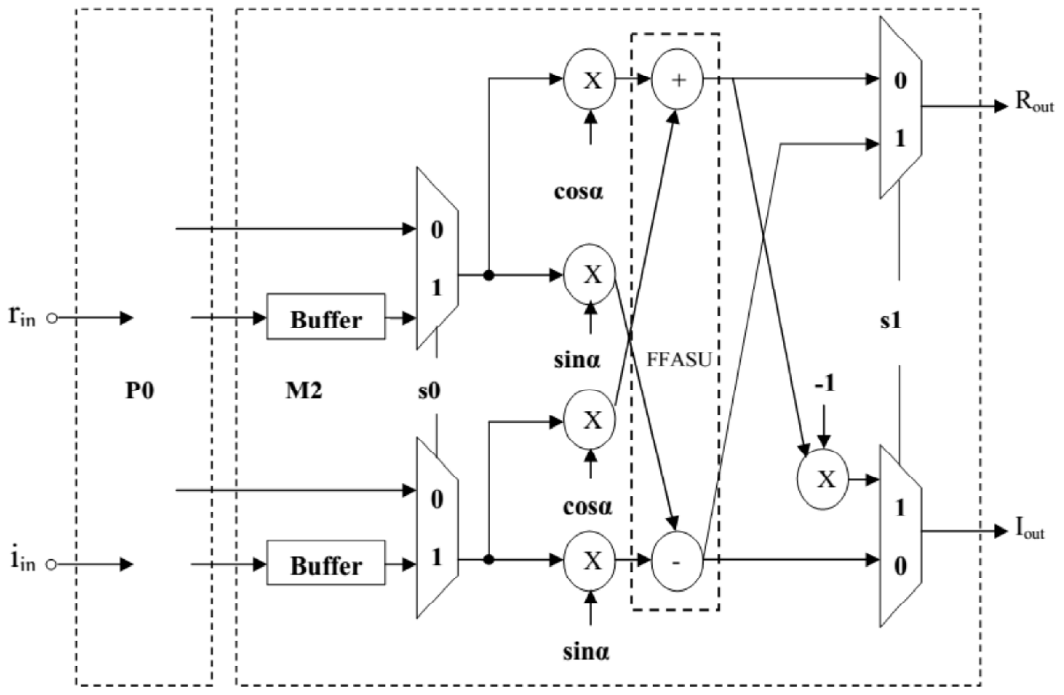


Figure 7: The Architecture of P3

The pipelined configuration of the P3 is illustrated in figure.7. "M2"is a sub-module employed to estimate twiddle factors multiplications consisting of four multipliers and two adders. One ROM is needed for storing twiddle factors. P3 is a fully functional DIF butterfly which extends a helping hand to arbitrary twiddle factor multiplications.

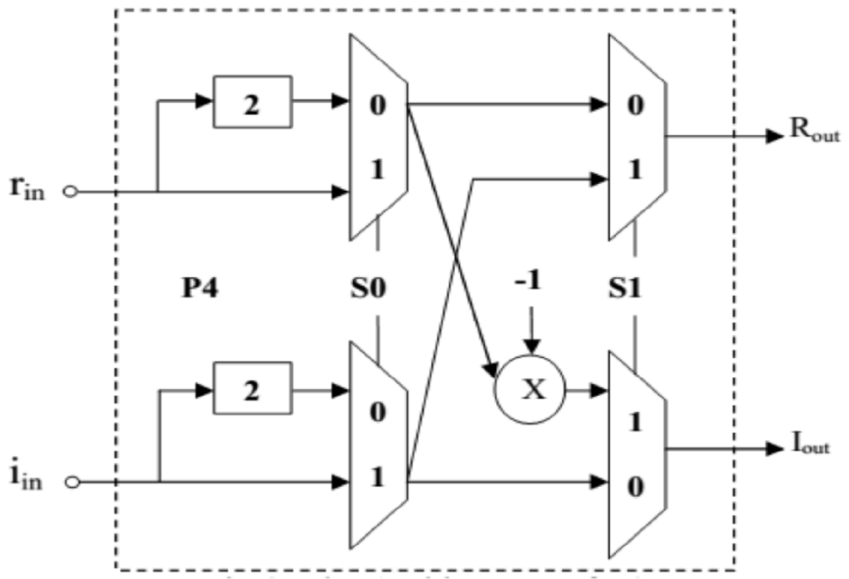


Figure 8: The Architecture of P4

P4 is illustrated in figure.8. It possesses no multipliers and adders. The task of P4 is to swap the authentic part and the imaginary part of complex data, and reverses the order of two consecutive complex data in the interregnum . These tasks are managed by "S0". Signal "S1" is employed to manage the multiplication by *–j*.

2. D DWT

DWT evaluates the data at various frequencies with varied time resolutions. Figure 9 illustrates the DWT disintegration of the image. The DWT decay integrates low-pass '*l*' and high-pass '*h*' filtering of the images in both horizontal and vertical directions. After each filtering, the output is down-sampled by two. Further disintegration is performed by applying the above procedure to the LL sub-band.

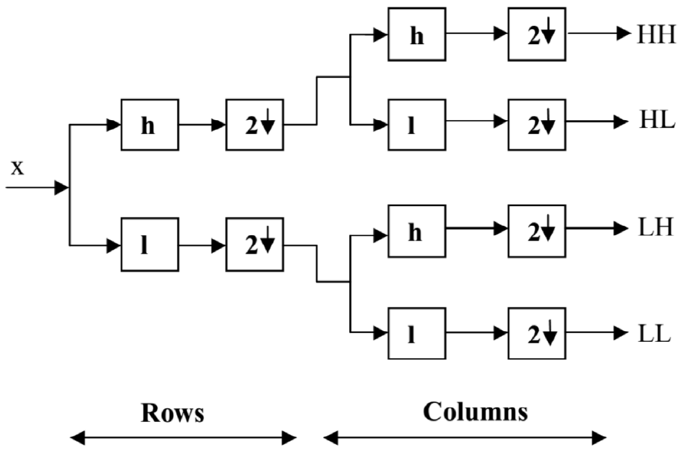


Figure 9: 2-D Discrete Wavelet Transform

For efficient implementation of 2D-DWT we have adopted Lifting based scheme in our proposed work. The Lifting scheme comprises three easy stages: 1.split, 2. Predict (P) and 3.update (U). In the split stage, the input sequence *x(n)* is separated into two disjoint set of samples, even indexed samples (even samples) $x_e(2n)$ and odd indexed samples (odd samples) $x_o(2n+1)$. In the predict stage, even samples are employed to forecast the odd samples in accordance with the correlation available in the signal. The divergences between the odd samples and the related predicted values are estimated and denoted as detailed or high-pass coefficients, *h (2n+1)*. The update stage employs the fundamental qualities of the coarser signals in other words; they possess the identical average value of the signal. In this stage, the coarse or low-pass coefficient *l (2n)* is derived by updating the even samples with detailed coefficient. The block diagram of the lifting based DWT is illustrated in figure 11.

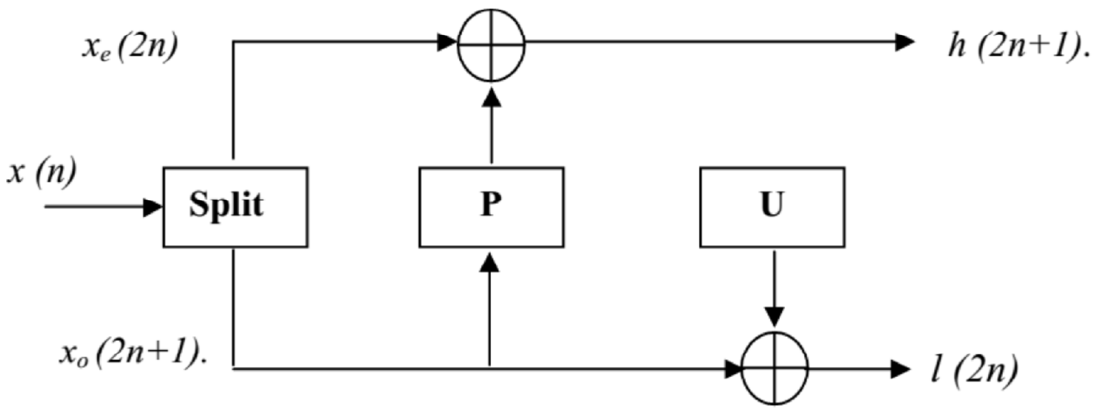


Figure 10: Lifting scheme Forward Dwt

The calculation of high-pass and low-pass coefficients for two consecutive values for wavelet is shown below

$$h(2n+1) = x_o(2n+1) - 0.5[x_e(2n)] + x_e(2n+2) \quad (6)$$

$$l(2n) = x_e(2n) + 0.25[h(2n-1)] + h(2n+1) \quad (7)$$

Where

x(n)	- Input Signal.
h (2n+1)	- High pass filter Coefficient or the Details coefficients .
l (2n)	- Low pass filter Coefficient or the approximations coefficients.

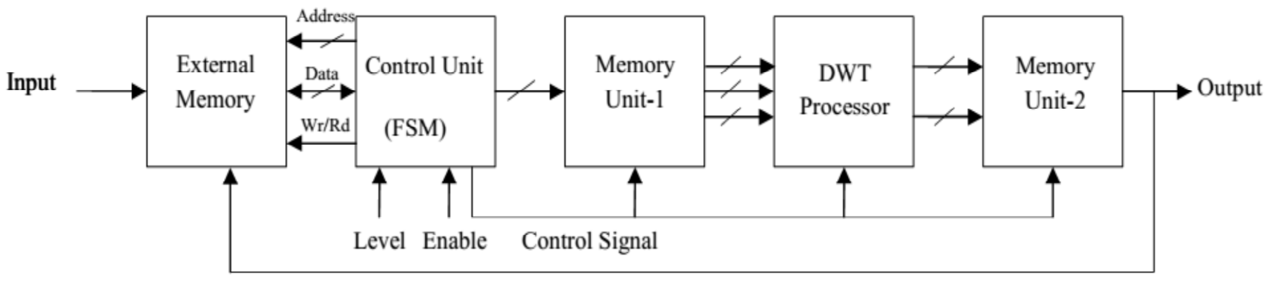


Figure 11: Lifting Based 2D-DWT Block diagram

Memory Unit (1 & 2)

The memory units are designed as cache memory to speed up the DWT process. The memory unit 1 acts as a split stage in lifting scheme. The block diagram of the Memory Unit-1 is shown in figure 12.

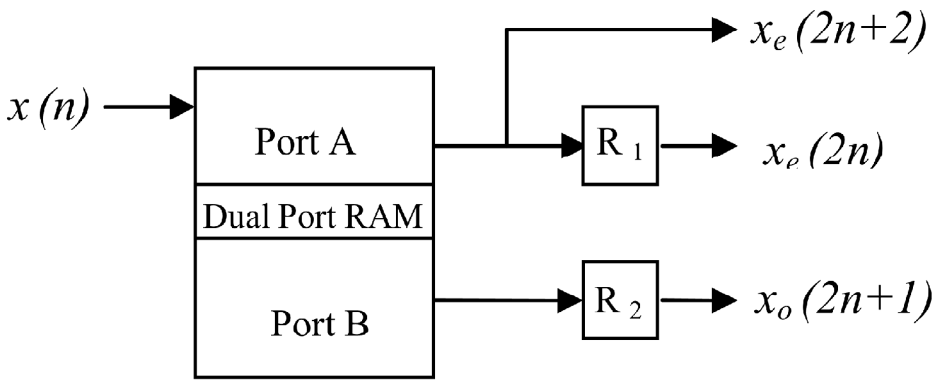


Figure 12: Block Diagram of Memory Unit-1

Memory unit-1 consists of a dual port RAM with one of its port (A) is connected to input signal *x(n)* and the other port (B) is without any connection. The even samples locations from the external memory are read and stored in the register R_1 through port A and similarly the add samples locations from the external memory are read and stored in the register R_2 through port B. This module delivers 3 outputs as shown in figure to the DWT processor for each clock pulse. The memory Unit-2 also consist of a dual port RAM with 2 input and 1 output used to bring out the low pass filter coefficient or the approximations coefficients for each level to the external level for next level of computation.

DWT Processor

The 2D DWT Processor represents the hardware design of (5/3) lifting scheme filter, this module performs the operation of Predict and Update stage in lifting scheme. The processor can be used for the computation of DWT coefficients along the row and column. Figure 13 exhibits the block diagram of the DWT Processor.

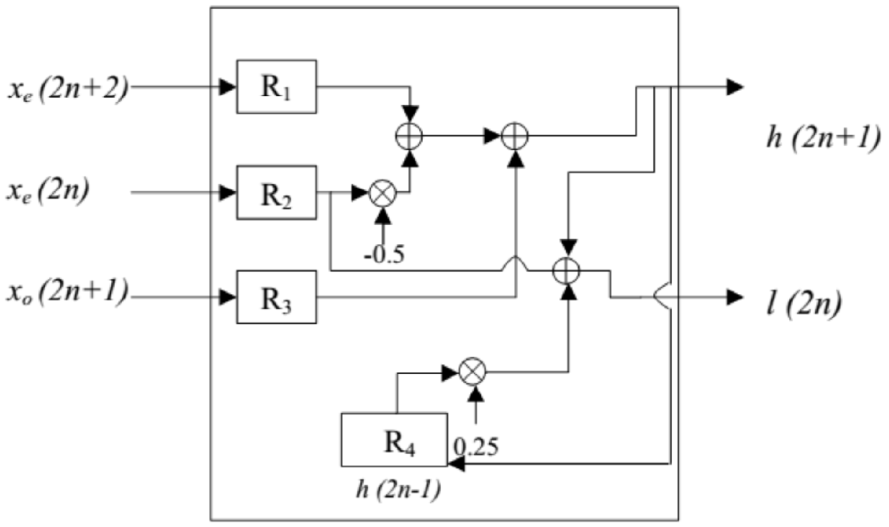


Figure 13: Block diagram of the DWT Processor

The DWT processor consists of four registers R_1, R_2, R_3 and R_4. The first three registers store the output data from the memory unit-1 and the register R_4 helps in storing the previously computed high pass coefficient value. The filter coefficient values -0.5 and 0.25 for (5/3) lifting scheme filter is stored in ROM and supplied when needed. The adders and Multipliers used here is of floating point type and has been discussed earlier in this paper.

Control Unit

The control unit in this 2D DWT architecture controls each section of the complete architecture for its proper functioning. The control unit is designed with a FSM in our technique. It provides control signals for Read, Write and Enable. It also generates the address of memory locations from where the data can be read or write. The 2 user inputs level and enable helps the user in selecting the level of decomposition and to enable the complete DWT process. This unit is mainly responsible for reading data from external memory, Enable DWT processor units operation and writing the sub-bands computed to the external memory.

Floating Point Multiplier

The IEEE 754 single precision floating point representation of real numbers in binary format is illustrated in figure 14.

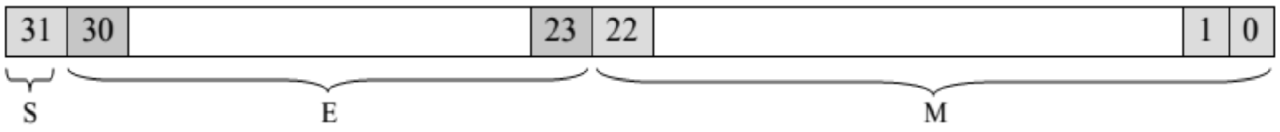


Figure 14: IEEE 754 single precision floating point representation

Normalized floating point numbers are represented by the expression:

$$Z = (-1^{s}) * 2^{(E\text{-}Bias)} * (1.\,M) \quad (8)$$

Where,

Bias for 32 bit =127

All the multiplication computation in our proposed architecture is implemented using the floating point multiplier. To multiply two floating point numbers the following is done:

- Multiply the significand; i.e. (1.M1*1.M2)
- Place the decimal point in the outcome
- Add the exponents; i.e. (E1 + E2 – *Bias*)
- Obtain the sign; i.e. s1 XOR s2
- Normalize the outcome by achieving 1 at the MSB of the outcome's significand
- Round off the outcome to fit in the available bits
- Check for underflow/overflow incidence

The hardware execution of the floating point multiplier is illustrated in figure 15.

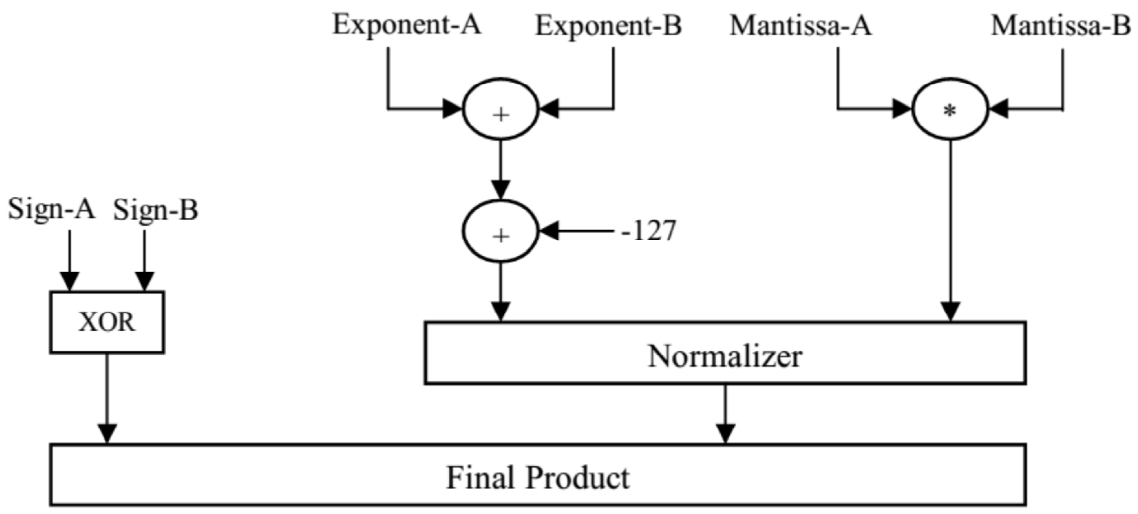


Figure 15: Architecture for Floating Point Multiplier

Fused Floating point Add-Subtract unit (FFASU)

Floating point addition and subtraction are the fundamental arithmetic functions discussed in this paper. In both, it is highly essential to see to it that both operands have identical exponent value. The architecture for the Fused Floating Point Add-Subtract Unit employed in this paper is shown in the figure 16. The various steps in the proposed algorithm are given below.

- Segregate the inputs A and B into Sign, Exponent and Mantissa respectively.
- The sign bits of both the inputs are XORed and stored in the output register.

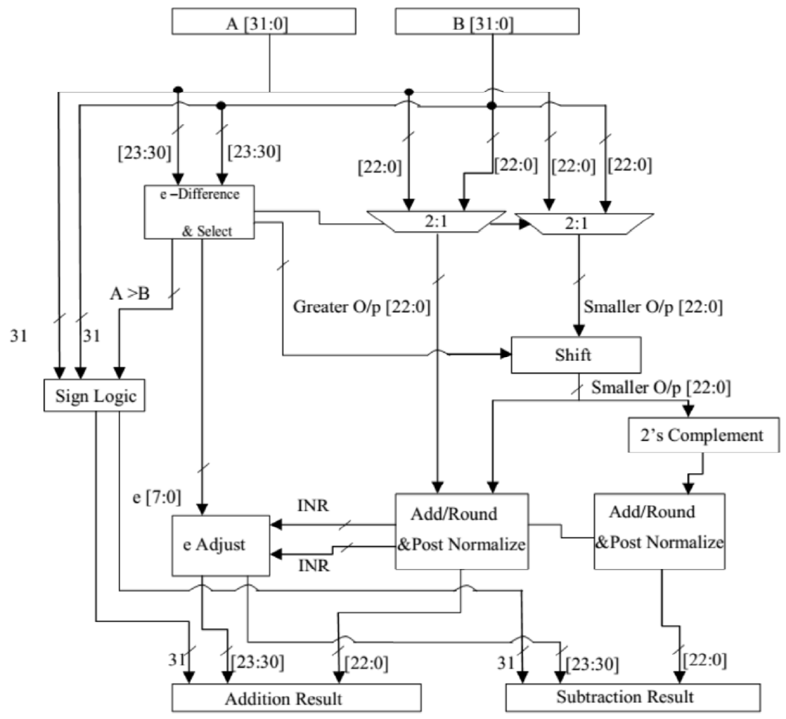


Figure 16: Architecture of Fused Floating point Add-Subtract unit

- Locate the difference in exponent values of both the input‘d’ and also find the greater and smaller values between the two.
- Choose the corresponding Mantissa of the smaller exponent and shift it right by‘d’ times.
- Increment the exponent by 1 for each shift operation.
- For addition the shifted mantissa of one of the input is added with the other mantissa and in case of subtraction 2’s complement is considered for shifted mantissa of one of the input and then added with the other mantissa.
- The rounding off and normalization operations are carried out in the next step.
- The final added and subtracted output of the input A and B are stored in the respective register.

Quantization

A quantizer is used to reduce the number of bits needed to store the transformed coefficients by reducing the precision of those values. As it is a many-to-one mapping, it is a lossy process and is the main source of compression. The 8×8 block of our HWT coefficients is then fed as input to the Quantization module. Quantization is done by dividing each HWT coefficient by corresponding quantizer value followed by rounding off the coefficient value to a nearest one.

$$f_q(x,y) = Round\left[\frac{f(x,y)}{Q(x,y)}\right] \tag{9}$$

The quantizer module incorporates ROM and divider. The ROM is helps in storing the quantizing values. The divider performs its operation in a pipelined manner. The initial HWT coefficient coming out from the HWT module is divided by the first value from the corresponding ROM (Quantization table values are already stored in ROM), and second HWT coefficient is divided by the second ROM value, like wise total 64 coefficients are divided separately by the values in ROM. A standard Quantization table is shown in table 2. The Quantization module is also implemented in pipeline architecture. Block Diagram of the implementation is shown in figure 17.

Table 2
A standard Quantization table

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	13	16	24	40	57	69	56
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

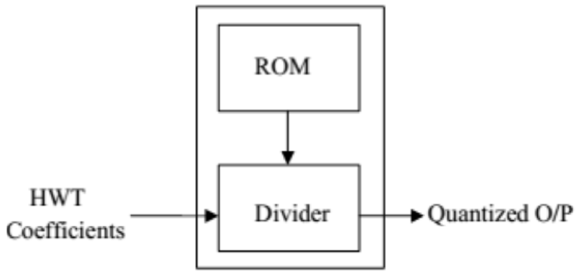


Figure 17: Architecture of Quantizer

Variable Length encoding

Variable Length Encoding (VLE) is the concluding phase of our image compression unit. The Quantized image can be more compressed by means of VLE. It integrates the following three steps:

1. Zigzag scanning
2. Run Length Encoding (RLE), and
3. Huffman coding.

The block diagram of VLE is drawn in figure 18.

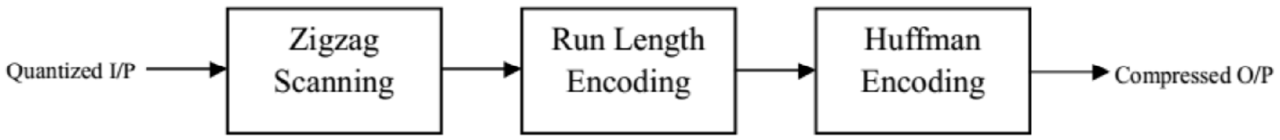


Figure 18: The block diagram of VLE

The quantized HWT coefficients gathered after executing the Discrete Cosine Transformation to 8×8 block of pixels they are furnished as input to the Variable Length Encoder (VLE). These quantized HWT coefficients will have non-zero low frequency components in the top left corner of the 8×8 block and higher frequency components in the residual places. The higher frequency components approximate to zero after quantization. The low frequency HWT coefficients are more significant than higher frequency HWT coefficients. Even if we overlook certain higher frequency coefficients, we are able to effectively rebuild the image from the low frequency coefficients only. The Zigzag Scanner block makes fullest use of this trait. In zigzag scanning, the quantized HWT coefficients are read out in a zigzag order, as exhibited in the figure 19. By organizing the coefficients in this way, RLE and Huffman coding is executed to enable added compression of the data. The scan places the high-frequency components jointly, which are, more often than not, found to be zeroes.

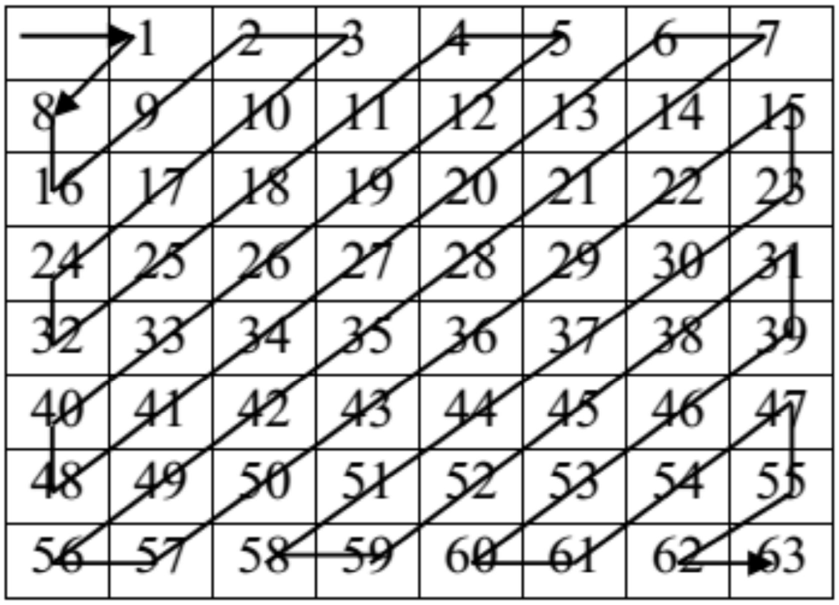


Figure 19: Zigzag Scan Order

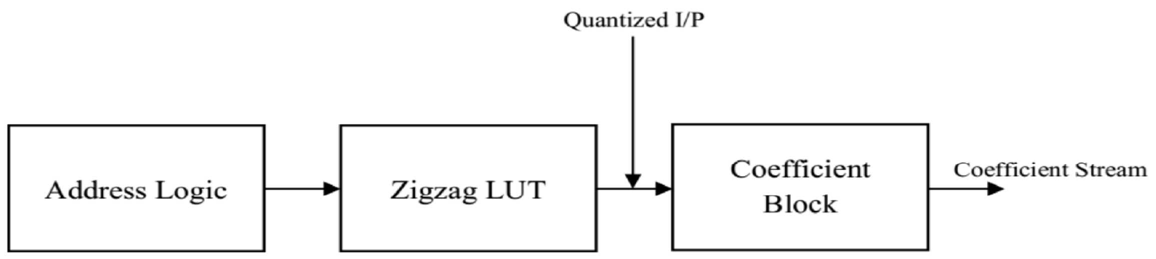


Figure 20: Block Diagram of Zigzag Scanning

The architecture for the zigzag buffer comprises a dual-port 128×8-bit memory block, segregated into two 64 entry buffers, employed in ping-pong mode. Quantized data is written to one buffer with the zigzag ordered data is read from the other buffer. After each 8×8 block, the task of the buffers is exchanged. The address logic comprises a counter. The zigzag lookup table (LUT) modulates the sequential address to the address within the coefficient block with the zigzag ordered coefficient. The LUT is performed by means of a memory block on the FPGA. The block diagram of the LUT based zigzag scanning procedure is well depicted in figure 20.

Run Length Encoding (RLE)

The quantized coefficients are read out in a zigzag order from DC module to the highest frequency module. RLE is employed to encode the string of data from the zigzag scanner. The conservative Run length encoder encodes the coefficients in the quantized block into a run length (or number of occurrences) and a level or amplitude, as for instance, communicates four coefficients of value “11” as: {11, 11, 11, 11}. By employing RLE, the level is 10 and the run of a value of 10 is four. By using RLE, {4,11} is communicated, decreasing the quantity of data communicated. Characteristically, RLE encodes a run of symbols into two bytes, a count and a symbol. By describing an 8 x 8 block overlooking RLE, 64 coefficients are employed. To compress the data more, several quantized coefficients in the8 x 8 block are made zero. Coding can be stopped in a situation where there are no more non-zero coefficients in the zigzag sequence. Figure 21 illustrates the architecture employed for Run Length Encoding procedure and a demonstration of run length encoding is exhibited in table 3.

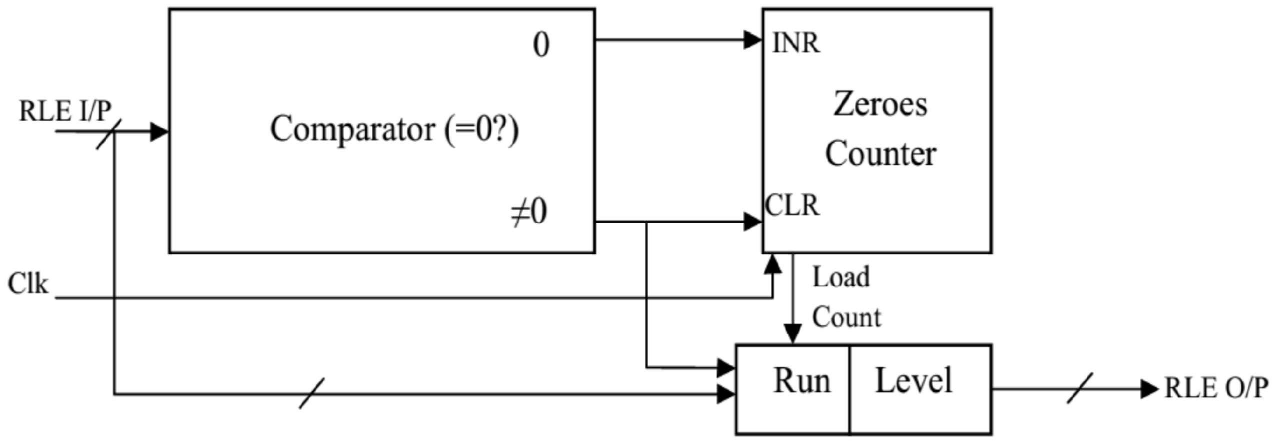


Figure 21: Architecture for Run Length Encoding

Table 3
Sample Run length encoding

Original Sequence	17	8	54	0	0	0	97	5	16	0	45	23	0	0	0	0	43
RLE Sequence	17	8	54	0	2	97	5	16	0	0	45	23	0	3	43		

But usually in a characteristic quantized HWT matrix the number of zeroes is greater in relation to nonzero coefficients being repetitive. Therefore, in our architecture for run-length encoder we make fullest fuse of this property of more number of zeroes in the HWT matrix. In the ambitious architecture the number of intermediate zeros in between non-zero HWT coefficients are counted in contrast with the conventional run-length encoder where number of incidences of repeated symbols are counted.

Huffman Encoding

After the RL encoding process the output of the RL encoder module is collected and fed to Huffman encoder for further compression of the image data. In Hoffman encoding process the input data with more frequency is assigned a short binary bit and that which has low frequency is assigned a long binary bit. Our implementation of Huffman encoder consists of three modules 1. Histogram 2.Sorting and 3.Coder. The histogram module first finds the frequency of occurrence of each element and passes it to the sorting module. The sorting module helps in arranging the frequencies in ascending order. In the last coder module the sorted data are coded with appropriate Huffman codes. The block diagram for the Huffman encoder is shown in figure 22.

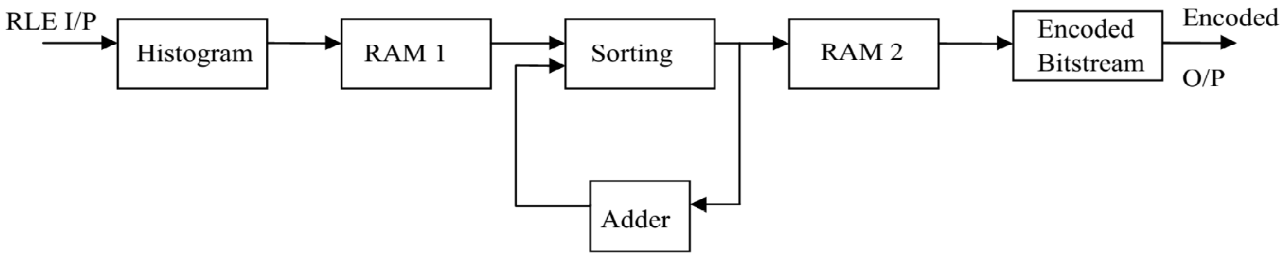


Figure 22: Block Diagram of Huffman Encoder

Steps to compute Huffman code:

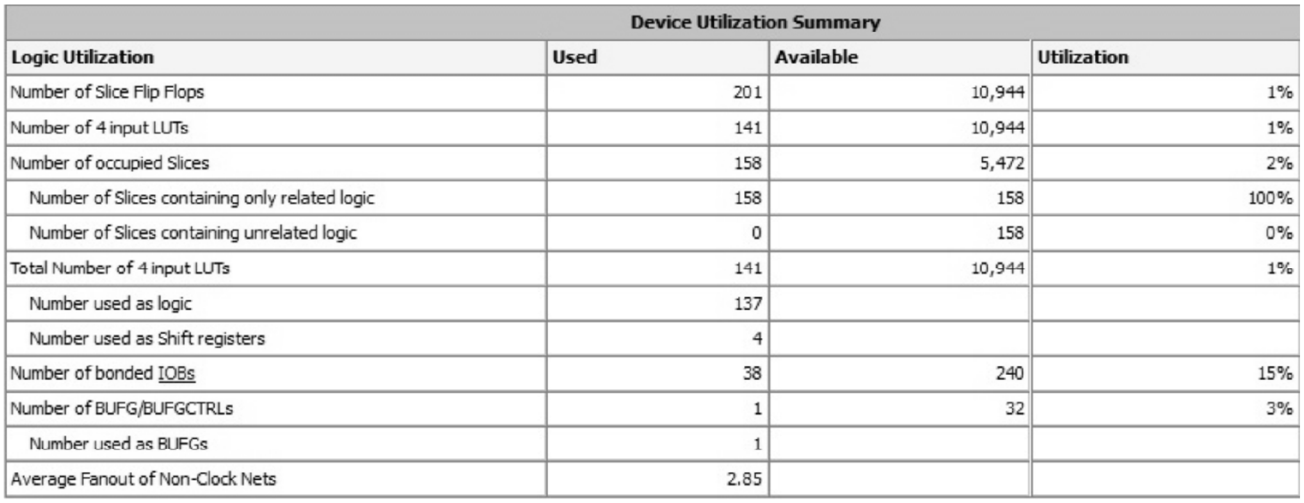
1. The frequencies are sorted in increasing order.
2. Select the last two least frequent values.
3. Form a binary tree by labeling its edges as ‘0’ or ‘1’.
4. Substitute the least two values with their sum.
5. Update the sequence
6. Again take the last two values and form a binary tree.
7. Go to step 2 until the last value.

4. EXPERIMENTAL RESULTS

The architecture for the proposed image compression technique is implemented in Xilinx ISC 14.1 using Verilog-HDL. MATLAB was used to find the input image pixels for our design. For testing the efficiency of our proposed design with the existing techniques we considered a “Lena” image with a standard pixel size of 256×256 as input image for compression. The image was read in MATLAB and the corresponding pixel value was noted and then fed to our proposed architecture as input. The observations and comparisons of the proposed image compression technique based on Area , Power consumption and performance was discussed separately in the following sub-sections.

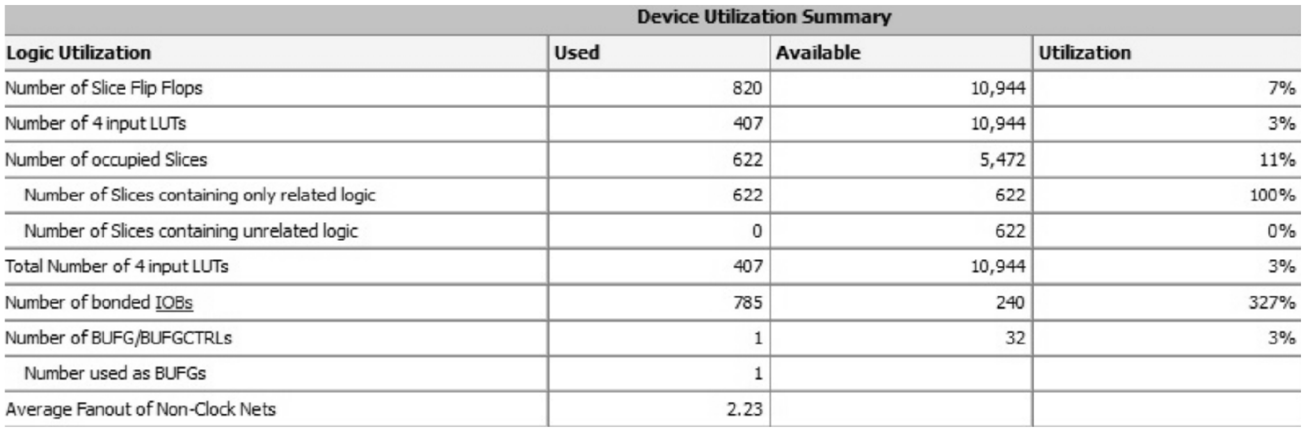
Area

The device utilization table for the HWT, Zig-Zag, RLE and Huffman coding modules are given below in figure.23, 24, 25 and 26 below.



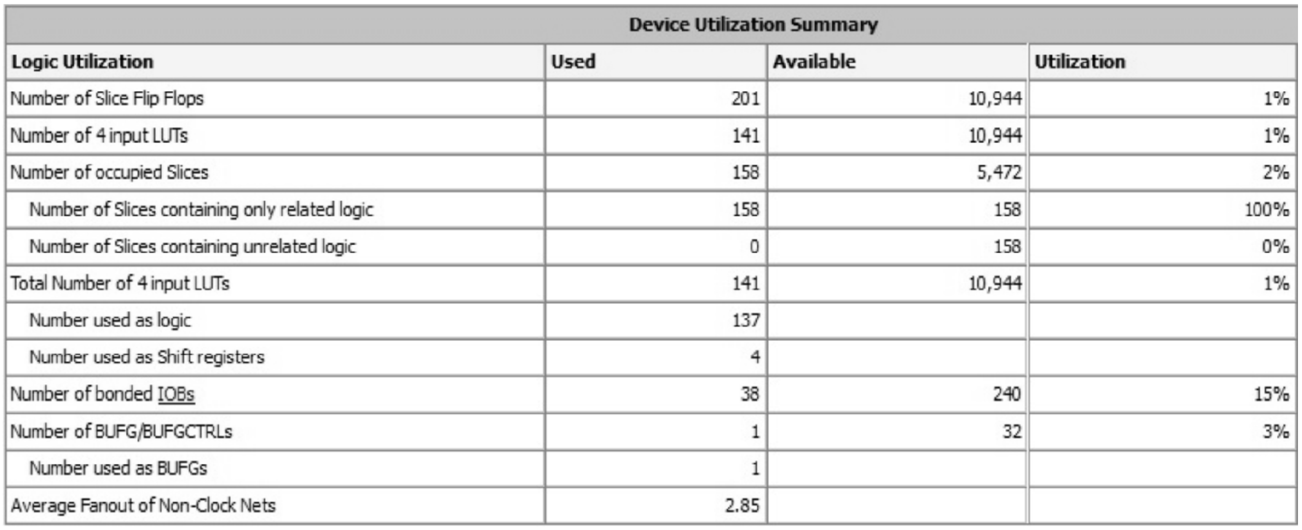
Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	201	10,944	1%
Number of 4 input LUTs	141	10,944	1%
Number of occupied Slices	158	5,472	2%
Number of Slices containing only related logic	158	158	100%
Number of Slices containing unrelated logic	0	158	0%
Total Number of 4 input LUTs	141	10,944	1%
Number used as logic	137		
Number used as Shift registers	4		
Number of bonded IOBs	38	240	15%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Average Fanout of Non-Clock Nets	2.85		

Figure 23: Device Utilization Summary for the Proposed HWT module



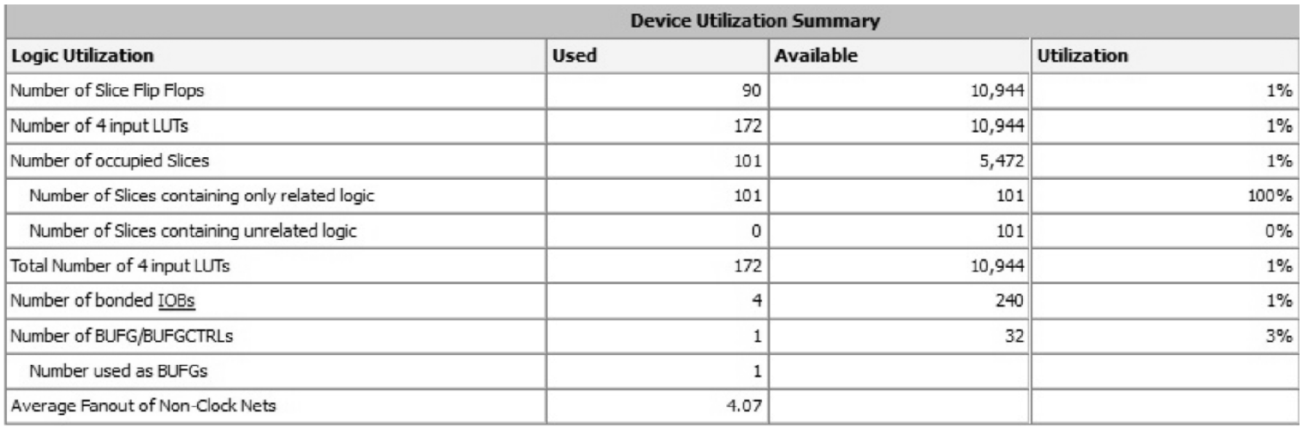
Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	820	10,944	7%
Number of 4 input LUTs	407	10,944	3%
Number of occupied Slices	622	5,472	11%
Number of Slices containing only related logic	622	622	100%
Number of Slices containing unrelated logic	0	622	0%
Total Number of 4 input LUTs	407	10,944	3%
Number of bonded IOBs	785	240	327%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Average Fanout of Non-Clock Nets	2.23		

Figure 24 : Device Utilization for the Zig-Zag scanner module



Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	201	10,944	1%
Number of 4 input LUTs	141	10,944	1%
Number of occupied Slices	158	5,472	2%
Number of Slices containing only related logic	158	158	100%
Number of Slices containing unrelated logic	0	158	0%
Total Number of 4 input LUTs	141	10,944	1%
Number used as logic	137		
Number used as Shift registers	4		
Number of bonded IOBs	38	240	15%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Average Fanout of Non-Clock Nets	2.85		

Figure 25: Device Utilization Summary for RLE module



Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	90	10,944	1%
Number of 4 input LUTs	172	10,944	1%
Number of occupied Slices	101	5,472	1%
Number of Slices containing only related logic	101	101	100%
Number of Slices containing unrelated logic	0	101	0%
Total Number of 4 input LUTs	172	10,944	1%
Number of bonded IOBs	4	240	1%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Average Fanout of Non-Clock Nets	4.07		

Figure 26: Device Utilization for the Huffman Encoding module

Table 4
Comparison of Device Utilization of different modules

Module	*No. of FFs*	*No. of LUTs*	*Total no. of occupied slices*
HWT	201	141	158
Zig-zag Scanner	820	407	622
RLE	201	141	158
Huffman Encoding	90	172	101

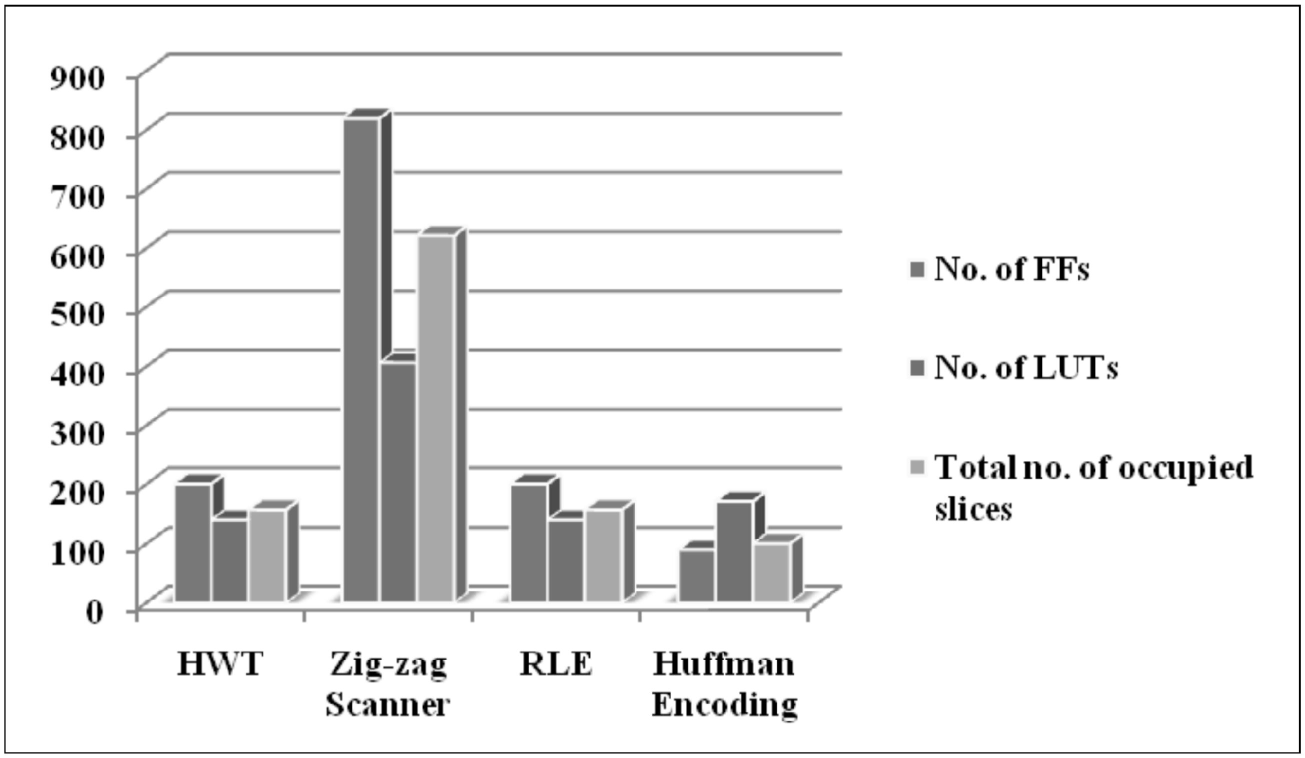


Figure 27: Graph showing device utilization by various module

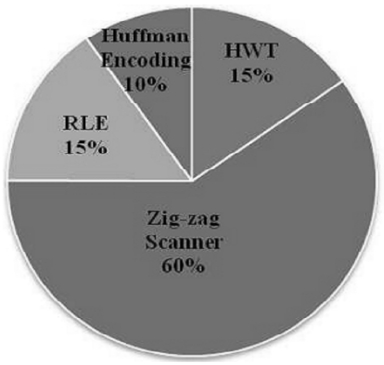


Figure 28: Pie-chart showing the contribution of each module in total area

Figure 28 exhibit Pie-chart for the contribution of each module in the total occupied area for our proposed image compression architecture. We can observe that major portion (60%) of the architecture is utilized by the Zig-Zag scanner because of the more number of memory usage in our design using LUT. Our proposed HWT and RLE utilized only 15% of the area usage and the Huffman encoding technique used here contribute only 105 to the total component utilization.

Power

The power consumed by each module in our proposed image compression architecture can be determined from the report generated by the XPower analyzer tool in Xillinx ISE. The power consumed by each module is compared with each other in table 5.

Table 5
Comparison of Power consumption by different modules

Module	*Dynamic (mW)*	*Quiescent (mW)*	*Total (mW)*
HWT	7	167	174
RLE	22	167	189
Huffman Encoding	20	167	187

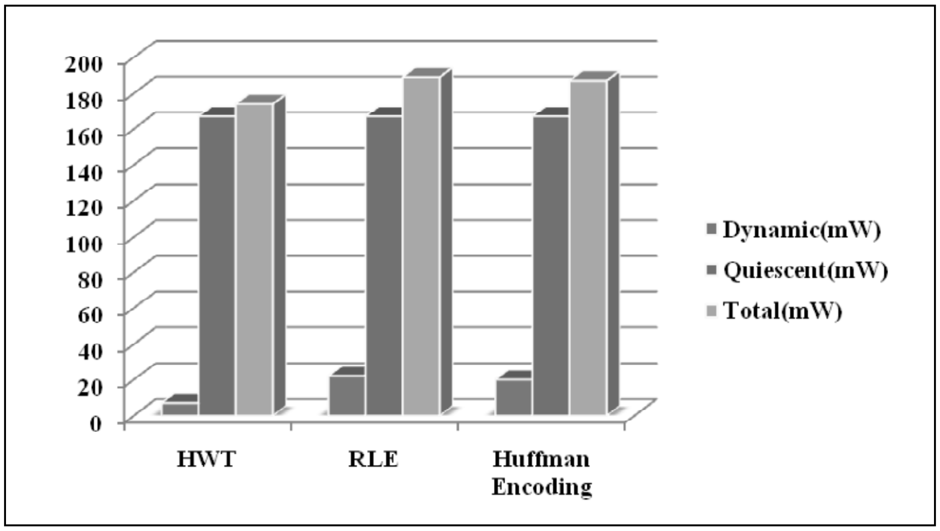


Figure 29: Graph showing power consumption by various module

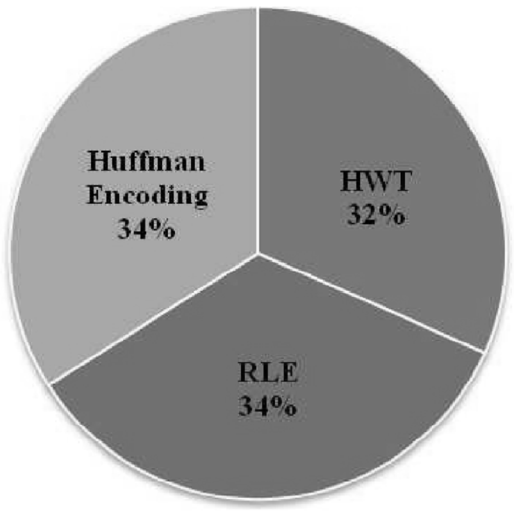


Figure 30: Pie-chart showing the contribution of each module in total power consumption

Table 6
Power Comparison table of floating point and conventional Modules

Modules		*Power (mW)*
	Conventional	176
Multiplier	Floating Point	199
	Conventional	321
HWT	Floating Point	174

Table 6 exhibits the comparison of power between the conventional and Floating point Multiplier and HWT modules. The result from the XPower analyzer shows that our design for Floating point HWT consumes less power than the conventional HWT architecture.

Performance

After processing the complete 65,536 ([8×8]*1024) pixels the output bit stream data are obtained for the compression ratio calculation as below,

$$\textit{Compression Ratio} = \frac{\textit{Uncompressed Size}}{\textit{Compressed Size}}$$

From our computation we obtained a compression ratio of 6:1 i.e.) by using our proposed technique the compressed file size is 6 times smaller than that of the original file size. About 83.37% of the disk space can be saved by adopting our proposed compression technique.

5. CONCLUSION

In this paper we have described a combination of an efficient architecture for image compression using a modified floating point HWT for wavelet conversion and a low power variable length encoding process for lossey image compression. The complete architecture is implemented in Xillinx ISE 14.1 using verilog-HDL. The proposed architecture was synthesized using RTL compiler and the Device Utilization Summary was generated for each module separately and their contribution for area include HWT-15%, Zig-zag scanner-60%, RLE-15% and Huffman Encoder-10% respectively . Experiments were conducted by using a standard 256×256 'Lena' image and a compression ratio of 6:1 was obtained. The power consumed by each the modules were estimated using the XPower analyzer tool in Xillinx ISE and found that all the modules used in the design consumed an average power of 33.33%.

REFERENCES

[1] Vijaya Prakash. A.M and K.S. Gurumurthy "VLSI Architecture for Low Power Variable Length Encoding and Decoding for Image Processing Applications, " International Journal of Advances in Engineering & Technology, Jan 2012.

[2] Devangkumar Umakant Shah and Chandresh H. Vithlani, "VLSI-Oriented Lossy Image Compression Approach Using DA-Based 2D-Discrete Wavelet", IAJIT, vol. 11, no.1, 2012.

[3] Soûa C. Olhede & Georgios Metikas, "The Hyperanalytic Wavelet Transform", Technical report, 2008.

[4] Wang Xu, Zhang Yan, Wang Fei and Ding Shunying, "A Configurable Floating-Point Discrete Hilbert Transform Processor for Accelerating the Calculation of Filter in Katsevich Formula." WSEAS Transactions on Communications, Vol. 11, no. 11, November 2012.

[5] Bhuyan M.S., Amin N., Madesa Md.A.H., and Islam Md.S., "FPGA Realization of Lifting Based Forward Discrete Wavelet Transform for JPEG 2000," International Journal of Circuits, Systems and Signal Processing, vol. 1, no. 2, pp.124-129, 2007.

[6] Calderbank A. R., "Wavelet Transforms That Map Integers to Integers," Applied and computational harmonic analysis, pp. 332-369, 1998.

[7] Chakrabarti C., Vishwanath M. and Owens R. M., "Architectures for Wavelet Transforms: A Survey," J. VLSI Signal Process, vol. 14, pp. 171-192, 1996.

[8] Davis G.M. and Nosratinia A., "Wavelet Based Image Coding: An Overview," Applied and Computational Control, Signals, and Circuits, 1998.

[9] Daubechies I. and Sweldens W., "Factoring Wavelet Transforms into Lifting Steps," J. Fourier Anal. Appl, vol. 4, pp. 247-269, 1998.

[10] Dhulap S.S. and Nalbalwar S.L., "Image Compression Based on IWT, IWPT & DPCM-IWPT," International Journal of Engineering Science and Technology, vol. 2, no. 12, pp. 7413-7422, 2010.

[11] Huang C.T., Tseng P.C. and Chen L.G., "Flipping Structure: an Efficient VLSI Architecture for Lifting-Based Discrete Wavelet Transform," IEEE Trans. Signal Process., vol. 52, no. 4, pp. 1080-1089, 2004.

[12] Huang C.T., Tseng P.C., and Chen L.G., "VLSI Architecture for Discrete Wavelet Transform Based on B-Spline Factorization," in Proc. IEEE Workshop Signal Process. Syst., pp. 346-350, 2003.

[13] Jeng Y.H., Hsu S.L. and Chang Y., "Entropy Improvement for Fractal Image Coder," International Arab Journal of Information Technology (IAJIT), vol. 9, no.5, September 2012.

[14] Kharate G. K., Ghatol A. A., and P.P. R., "Image Compression Using Wavelet Packet Tree" ICGST-GVIP Journal, vol. 5, no. 7, 2005.

[15] Lin C., Zhang B. and Zheng Y.F., "Packed Integer Wavelet Transform Constructed by Lifting Scheme," IEEE Transactions on Circuits And Systems For Video Technology, vol.10, no.8, 2000.

[16] Lin Z., Hoffman M.W., Schemm N., Leon-Salas W. D. and Balkir S., "A CMOS Image Sensor forMulti-Level Focal Plane Image Decomposition,"IEEE Trans. Circuits Syst. I, Reg. Papers, vol.55, no. 9, pp. 2561-2572, Oct. 2008.

[17] Maamoun M., Namane A., Neggazi M., Beguenane R., Meraghni A. and Berkani D., "VLSI Design for High-Speed Image Computing Using Fast Convolution Based Discrete Wavelet Transform," Proceedings of the World Congress on Engineering, vol. I, London, U.K, July 1-3, 2009.

[18] Mallat S.G., "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation," IEEE Trans. Pat. Anal. Mach. Intell, vol. 11, no. 7, pp. 674-693, 1989.

[19] Munteanu A., Cornelis J., and Cristea P., "Wavelet Based Lossless Compression of Coronary Angiographic Images," IEEE transactions on medical imaging, vol. 18, no. 3, 1999.

[20] Nebout C.L., Moury G., and Blamont J.E., "Status of onboard Image Compression for CNES Space Missions," Proc. SPIE'99 3808, pp. 242-256, 1999.

[21] Pujar J.H. and Kadlaskar L.M, "A New Lossless Method of Image Compression and Decompression Using Huffman Coding Techniques," Journal of Theoretical and Applied Information Technology, vol. 15, no.1, 2010.

[22] Rao C.H. and Latha M.M., "A Novel VLSI Architecture of Hybrid Image Compression Model Based on Reversible Blockade Transform," World Academy of Science, Engineering and Technology, vol. 52, 2009.

[23] Song M.S., "Entropy Encoding in Wavelet Image Compression," Representations, Wavelets and Frames A Celebration of the Mathematical Work of Lawrence Baggett (Editors Palle E.T. Jorgensen, Kathy D. Merrill and Judith A. Packer)," pp. 293-311, Springer, 2007.

[24] Sweldens W., "The Lifting Scheme: A Custom Design Construction of Biorthogonal Wavelets," Appl. Comput. Harmon. Anal, vol. 3, no. 15, pp. 186-200, 1996.

[25] Liu, Kai, Evgeniy Belyaev, and Jie Guo., "VLSI Architecture of Arithmetic Coder Used in SPIHT", International Journal of IEEE Transactions On Very Large Scale Integration (VLSI) Systems, Vol. 20, No. 4, April 2012.

[26] Sunil W. Bhise., "Pipelined Architecture of 2D-DCT, Quantization and Zigzag Process for JPEG Image Compression using Verilog", International Journal of Computer Science and Information technology, Vol.1, 2013.