CHAPTER 5

FPGA BASED HYPERSPECTRAL IMAGE COMPRESSION USING DWT-DCT AND SVD WITH PCA

5.1 INTRODUCTION

An image in its unique representation transmits vast amount of data and needs large volume of memory for storage. Image compression is a significant area in image processing, which effectively eradicates the visually irrelevant data. Generally, compressed images are directed over limited bandwidth channel with certain extra processing for robust (error free) transmission. Hyperspectral image compression algorithm is a most preferred choice, which contains image transform (in non-overlapping blocks), quantization of transformed coefficients and entropy coding. The JPEG image compression uses the 8x8 block-wise two dimension, Discrete Wavelet Transformation (2D-DWT) and Discrete Cosine Transform (DCT)

Images compressed by this standard are used globally. This algorithm provides the user to choose between amount of compression and quality as per the requirement of the image in different applications. The variable amount of compression makes this algorithm very much suitable for the transmission purpose as the user can adjust the bit rate of the transmission according to channel capacity. In image compression, most of the compression operation is done by using algorithm, which is a step by step process and implemented in software. When this data are passed to other system, hardware implementation is essential for data transmission process.
Later, at the reconstruction stage, the compressed data are received and again decompressed to form reconstructed image. Major existing work in image compression is carried out in software and hardware utilized for transformation of data to other systems. The system considers the external system to compute pixel variation and convert these values to serial values for calculation process. This system relies on other computing system to effectively compress the image and reconstruct it. In order to merge the software and hardware technology in image processing, Field Programmable Gate Arrays (FPGA) based architecture is implemented to embed the coding in hardware architectural design. Because of reprogrammable gate wise logic structure in FPGA it is easy to compute the pixel intensity in bit frame.

By using this process, an image compression can be done without referring external PC. The software design has large number of components, the critical path delay is high and the wire length for each component is large. So, in our proposed system, the software is made in a way by considering the above mentioned issues. The suggested FPGA when compared to the existing systems produced an optimal and simple transmission and reconstruction of compressed data on the other systems. In the FPGA based image processing system, there exists many circuits, which asses the pixel variation by logical architecture.

Generally, there occurs number of hardware development systems for image compression to calculate the variation in pixel intensity value with minimum power consumption and area reduction. In our research work, the image compression technique is implemented in the FPGA architecture. Compression architecture that consumes less area and power is deployed using the Very High speed integrated circuit hardware Description Language (VHDL).
In our proposed work, wavelet based transformation is done, which plays an important role in image processing applications. The advantages of the Wavelet based compression are robustness even in the presence of transmission and decoding faults, optimal frequency resolution at lower frequencies, optimal time resolution at higher frequencies and prevention against blocking artifacts.

Figure 5.1 Flow diagram of DWT-DCT-JPEG

The main advantage of wavelets over DCT are absence of blocking artifacts and the multi-scale nature of the DWT which allows near-optimal compression of features with a variety of different scales or sizes. The sub band coding based DWT resulted in heavy computation of wavelet transform. Compared to the other transforms, the DWT consumed less computation time and resources. The implementation of the DWT is also found to be very simple. The decomposition into sub-bands are highly
flexible in terms of resolution scalability. Figure 4.1 shows the flow diagram of DWT-DCT.

But DWT results low compression rates, it can’t individually decompose the HSI with good compression ratio, so it integrates the DCT technique to decompose the HSI into one more level for better compression ratio. The merits of DCT include flexibility at each block, parallel processing, minimal memory cost, etc. The approximation of the signals that has less number of coefficients are improved using the DCT. These techniques are simulated in the FPGA area and power reduction is done by using DWT with DCT combination of FPGA architecture with Run Length Encoding (RLE) encoder for image compression technique.

5.2 FPGA ARCHITECTURE

FPGA are pre-fabricated silicon devices, which can be electrically programmed to develop nearly any type of digital circuit or system. They offer a number of fascinating benefits over fixed-function Application Specific Integrated Circuit (ASIC) tools such as standard cell. The two vital technologies, which differentiates FPGAs are, architecture and Computer Aided Design (CAD) tools. These tools enable the user to utilize a FPGA designs. It also provides cheaper result and quicker time to market as compared to ASIC, which generally need a lot of resources in terms of time and money to acquire chief device.

On the other hand, FPGAs consumes less than a minute to configure and they cost about hundred dollars to a few thousand dollars. The requirement of user changes and for continuously changing requirements, a part of FPGA can be partially reconfigured whereas the rest of an FPGA is still running. Any further updates in the ultimate product can be simply
improved by downloading a new application called bit stream. Normally FPGAs consists of the following functions.

- Programmable logic blocks, which establish logic functions.
- Programmable routing that joins these logic functions.
- I/O blocks that are associated to logic blocks via routing interconnect, which make off-chip connections.

**Figure 5.2 Overview of CLB routing**

A generalized example of FPGA is illustrated in Figure 5.2. In the figure, Configurable Logic Blocks (CLBs) are organized in a two dimensional grid and are connected by programmable routing resources. In FPGA, I/O blocks are placed at the periphery of the grid and these blocks are linked through the programmable routing interconnect. In FPGA, the programmable or reconfigurable term denotes their capability to establish a new parameter on the chip once the fabrication of FPGA is complete. FPGA’s reconfigurability / programmability depend on a fundamental programming
technology, which leads to a change in the performance of a pre-fabricated chip after its fabrication. In this research work, the model of FPGA used is SPARTAN-6 X.

5.2.1 Configurable Logic Blocks (CLBs)

The sequential and the combinatorial circuits use the CLBs as their main logic resources. Each CLB element is interconnected to a switch matrix for accessing the common routing matrix.

A CLB element has pair of slices and these slices are arranged as a column. In CLB, the slice in the bottom of the CLB is denoted as SLICE (0) and the slice present in the top of the CLB is named as SLICE (1). The elements of CLB are depicted in Figure 5.3.

Figure 5.3 Illustrations of CLB elements
5.2.2 SPARTAN

Generally, the SPARTAN-6 FPGA family has two fields with optimized sub families. It also comprises of mixed feature corresponding to stringent market necessity for price-sensitive and high-volume applications. The personalized configuration data present in the Static Random Access Memory (SRAM-type) internal latches are stored and maintained by the SPARTAN. The configuration of SRAM varies from 3 Mb to 33Mb. The number of configuration is influenced by the size of the device and user scheme implementation options. The Configuration information stored in SRAM is loaded every time the FPGA is deployed.

By pulling the PROGRAM_B pin low, the configuration information can be reloaded whenever it’s necessary. Some methods and data layouts for loading the configuration are accessible. In SPARTAN-6, the bit-serial configurations has two modes, namely, master serial mode and slave serial mode. The FPGA produce the configuration clock (CCLK) signal during the master serial mode. The external configuration data source clock the FPGA during the slave serial mode. Normally, the Select MAP mode produce the CCLK signal for the byte-wide configurations. The CCLK signal for the 8-bit and 16-bit-wide transfer is received by the slave Select MAP mode.

The initial bit streams shift the clocking source to an external clock during the master serial mode. When compared to the internal clock, the process of switching is more quick and precise in the external clock. The boundary scan protocols based Joint Test Action Group (JTAG) pins are used to load the bit serial configuration data. The Serial Peripheral Interface (SPI) serial flash PROM that has industry standard when connected to the Spartan-6 FPGA is used to configure itself. The Spartan-6 FPGA when neither associated to the industry standard parallel NOR flash configures itself
through the BPI. Normally, the XC6SLX4, XC6SLX25, and XC6SLX25T do not support the BPI configuration. Also, the integration of Spartan-FPGA in TQG144 and CPG196 packages do not include the Byte Peripheral Interface (BPI).

The storage of multiple bit streams of FPGA configuration in single configuration source makes the Spartan-6 FPGAs to support the multi boot configuration. The proposed FPGA determines when and which configuration to be loaded. The Spartan- 6 FPGA provides anti-cloning designs, IP protection, etc. by integrating with the distinct factory-programmed device named, DNA identifier. Figure 5.4 shows the flow diagram of data transmission in FPGA.

![Flow of data transmission in FPGA](image)

**Figure 5.4 Flow of data transmission in FPGA**

### 5.3 OVERALL ARCHITECTURE OF SYSTEM DESIGN

Enormous data can be obtained from an image. Simulating the proposed architecture for diverse images in different precisions is a time
consuming process. The data samples that are obtained by implementing the suggested architecture in MATLAB are compared with the DCT output obtained by simulating the VHDL in FPGA. From the comparison it is found that, the MATLAB model provides optimal design. The standard test images are DCT transformed by the MATLAB model. With the use of the MATLAB double precision floating point data, the DCT (IDCT) reconstructs the images.

In our research work image to file conversion is done by using the MATLAB. The basic steps involved in the overall system are depicted in Figure 5.5.

![Figure 5.5 Steps involved in proposed system](image)

The overall system architecture is shown in figure 5.6. Initially, the original image is loaded into bit sequence convertor, where the bit sequence convertor converts the input image pixel into sequence of bit streams. Later, in DWT the bit stream is divided into low order and high order coefficients by using wavelet coefficient buffer. After dividing the low order and high order coefficient, convolution operation is performed, where the updation of DCT is performed. The updated streams are forwarded to the quantization process for performing the round off operation.

The rounded values are stored in a buffer called First In First Out (FIFO) buffer, from where it is transferred to the RLE counter and RLE encoder. RLE counter checks for the previous values if the values are same
then the value of the count are incremented. If the values are different the counter value is assigned to 1. The RLE encoding compress the bit streams of the data values that are present in multiple successive data elements. The output from the RLE encoder and counter is given to the multiplexer. Appropriate data is selected in the multiplexer.

Figure 5.6 Overall system architecture

The selected data are gain stored in a Read FIFO buffer and finally, the bit streams are reconstructed into original image. In our system, power management is established, in order to reduce the number of components in
FPGA architecture, to minimize the wire length and to decrease the area consumption. Configuration and control block efficiently control the wavelet buffer, DCT weight updation, Quantization, FIFO buffer, RLE counter and register. The implementation of these two blocks, namely, power manages and configuration control improves the performance of the proposed system. The proposed FPGA layout is shown in Figure 5.7.

Figure 5.7 Layout design of FPGA

5.4 2D-DWT FPGA Pipelined Architecture

The 2D-DWT based FPGA pipelined architecture is divided in to 1D-DWT for vertical and horizontal transforms. This architecture contains the components, such as, control unit, finite state machine, and internal memory block. By transferring all rows to the FPGA through the Peripheral Component Interconnect bus, the sub images are processed. The pipelining architecture is used to process the sub images in the horizontal 1D-DWT unit. The calculated coefficients are maintained in the internal memory.
The single port RAM stores the coefficients that are analogous to the sub image rows. The vertical 1D-DWT unit performs the vertical transformation level. The control unit organizes these stages to practice the entire sub image for producing enable signals, address lines, and so on.

![Figure 5.8 Block diagram of the pipelined DWT](image)

**Figure 5.8 Block diagram of the pipelined DWT**

At the final phase, the sub images that are wavelet transformed are accessed in the internal RAM. At this scheme, an Embedded Zero tree Wavelet (EZW) algorithm is used to represent the multi-scale sub images. Every important boundary information is involved in the computation, the block artifacts are prevented by the following quantization. The block diagram of the pipelined DWT is shown in Figure 5.8.

### 5.4.1 Horizontal DWT

In the horizontal 2D-DWT, all the four horizontal transformation levels are computed. As the intermediary results need not be transferred, the horizontal DWT permits the pipelined approach. 16 rows of sub images that are under the deliberation are applied the entire horizontal transform.
30 rows of northern sub image and 15 rows of southern sub image are transformed using the DWT transformation. The additional computations that are required by the vertical DWT are discussed further. Figure 5.9 shows the implementation of horizontal 1D-DWT unit.

![Figure 5.9 Four level horizontal 1D-DWT unit](image)

Each clock cycle of 1D-DWT unit considers four pixels from a row and demands 4 levels of horizontal transforms. Figure 5.9 show that the 1D-DWT unit contains four pipelined stages. Each transforms level use a pipelined stage. As each level of transform down samples the number of coefficients to half, the data throughput is maximum in the first stage. When compared to the other levels, the first and second level of the proposed module produce maximum throughput. At a clock cycle, the outputs of the first and second stage produce four coefficients. Among them, two coefficients have low frequency and the other two have high frequency.
After the initial stage, the high frequency and low frequency coefficients are joined together to get an uninterrupted stream of output. The simple delay element of the 2to4 module is used to transform the results of the first stage as input to the second stage. The results of the first stage are provided as four pixel wide bus to the second stage. The working style of the second stage is similar to the first stage, but the speed of the second stage is less when compared to the first stage. The second level low frequency and high frequency coefficients are combined into a single data stream and given an input to the third stage. At each clock cycle, the transform units of the three and four stages consume only one coefficient as input from the previous level and provide a low or high frequency coefficient as output. A pixel row of length 16 for 32 input clock cycles is processed by the horizontal 1D-DWT unit. The pixel value also includes the boundary pixel.

5.4.2 The w-bit Input 1D-DWT Unit (1i2o)

The 1-level DWT is implemented by the lifting method, which also includes the steps, such as,

- Fragment the input at the odd and even positions into coefficients
- Implement a predict-step operation
- Perform an update-step operation

The results of the predict-step operation and update-step operation are illustrated in Figure 5.10 and Figure 5.11. The addition, subtraction and shifts are the major components of the computation.
Using the suggested arrangement, the combinational functions among two flip-flops are accommodated within one column of the CLB. The Logiblox generates and configures the adder/subtractor that is registered. Here, the carry logic performs the carry logic operation. Based on the lifting schemes, the entire w-bit 1D-DWT unit is developed.

5.4.3 Internal Coefficient Memory

16×16 coefficients can be stored in the internal memory of the wavelet coefficient. As the bit widths of the coefficients are different from their sub bands, the memory block is sliced into 5 partitions. Figure 5.12(a)
shows the minimal affordable bit width for each sub bands and slices of memory are depicted. The structure of the internal memory is illustrated in Figure 5.12(b).

![Diagram](image)

Figure 5.12 Structure of the internal memory to store the wavelet coefficients (a) Minimum affordable memory bit width (b) Partitions of memory

### 5.5 DWT ARCHITECTURE IN PROPOSED FPGA

The proposed architecture includes carry look ahead adder, multiplexers, delay and registers. The architecture has two stages, where the first stage involves level shifter and the second stage has delay shift registers. The input given is in the form of 8-bit stream, later the bit stream is passed to the adder and a feedback is given from level shift registers to adder. The adder performs addition operation and transfers the output to multiplexer 1. Generally, multiplexer has two conditions, namely, set or reset condition and selected data condition. Multiplexer result is passed to delay, where it performs one step delay and forward the output to level shift registers. The proposed architecture of DWT system is implemented in Figure 5.13.
Figure 5.13 Architecture for the proposed DWT system

The level shift registers has the 16X16 matrix coefficients, which are called as weighted value. These values are stored in the internal memory and help to identify whether the stage is high level or low level filters. At the same time, output from the delay is passed to multiplexer 2 and multiplexer 3. The multiplexers 2 and 3 analyze the input of the adder and output obtained from delay in order to select the data. After the analyzing process, the selected data is passed to the multiplexer 1, which is known as predefined set of data. Whenever the data has to be selected by the multiplexer, it checks for the predefined condition. These operations are carried out in the first stage, which is based on the high pass filter.

In the second stage, the input stream is passed to the delay where it performs one step delay and then forwards the streams to adder. Here, a feedback loop from the level shifter is given as another input to the adder. The adder adds both the bits and sends to the multiplexer 4. The multiplexer compares the bits obtained from adder with the bits obtained in multiplexer 3.
And finally selects the data and direct it to the delay. After performing one step delay, the bit streams are forwarded to the multiplexer array. It consists of 8 numbers of arrays, which has smaller number of gates or transistors. The result attained in the multiplexer is given to the delay level shifter, where it performs one step delay and updates the weighted values. The weighted values acquired from the delay level shifter and level shifter is transferred to the multiplexer 5. It selects the essential data and the chosen data are stored in the output registers in the 8-bit format.

5.5.1 State Flow Diagram for DWT

The state flow diagram of the suggested DWT system is depicted in Figure 5.14. The figure explains the step by step process involved in the DWT transformation.

Algorithm

Enable CLK and RESET option

STEP 1: Load single input pixel from file

STEP 2: The input pixels are transformed into binary bit stream

STEP 3: If S_DWT=1

STEP 4: Then Ready state is assigned 1

The result of DWT enable is assigned 0

STEP 5: Then Ready state is set to 0

Check for the level= S_DWTW

The loop is given to STEP 3.

STEP 6: ElseIf S_DWT_W=1
STEP 7: Check for Enable, if Enable=0

STEP 8: Check for Level= S_DWT1

The loop is given back to STEP 6.

Else

STEP 9: Convolution operation is performed in Register 1, 2

STEP 10: Assign Llr=Reg 1 and Lhr=Reg 2

STEP 11: Again Convolution coefficient is assigned to Register 3,4

STEP 12: Assign Hlr=Reg 3 and Hhr = Reg 4

STEP 13: Level is updated

STEP 14: The same process is repeated for another pixel.

5.5.2 Detailed Processing of DWT in FPGA

Input:

Clk, enable, lengthd, dwtr.

Output:

Dwtr, dwtadden, LowH, HighH, HighL, LowL, dwt_w, dwt_c

Processing:

At raising edge of Clk,

lengthd = ‘111’

Check enable=1

If enable,

When State_DWT = 1, then
Update ready = 1;

Inp_data = ‘01111001’;

Update level = State_DWT_W;

When State_DWT_W = 1, then

Set level = State_DWT_I;

When State_DWT_I = 1, then

coeff1 = ‘00001000’;

Reg1 = conv(Inp_data and coeff1) = ‘01000000’;

coeff2 = ‘00000111’;

Reg2 = conv(Inp_data and coeff2) = ‘01000000’;

coeff3 = ‘00000001’;

Reg3 = conv(Inp_data and coeff3) = ‘01001100’

coeff4 = ‘00001001’;

Reg4 = conv(Inp_data and coeff4) = ‘00100011’;

Update level = State_DWT_L

When State_DWT_L = 1, then

If reglevel<lengthd

reglevel = reglevel + 1;

Lowlr(address) = 1;

Lowhr(address+1 to end) = 0;

Highlr(address) = 1;

Highhr(address+1 to end) = 0;
Update level = State_DWT_SD;

Else

Update level = next level;

When level = State_DWT_SD, then

HLr = HLr + Highlr + shift_right(reg1, reglevel) = ‘10100000’;
HHr = HHr + Highhr + shift_right(reg2, reglevel) = ‘10001000’;

Update level = State_DWT_SDI;

When level = State_DWT_SDI, then

LLr = LLr + Lowlr + shift_right(reg3, reglevel) = ‘10000000’;
LHr = LHr + Lowhr + shift_right(reg4, reglevel) = ‘10110010’;

Update level = Next level;

Figure 5.14 State flow diagram for the proposed DWT system
5.6 TRADITIONAL DCT ARCHITECTURE

A DCT coefficient has the components as shown in Figure 5.15 along with the clock latency of each component. The sum of all the eight columns is required, which is done by the accumulator. The column values from the data buffer are gathered by the accumulator. The accumulation operation consumes 64 clock latency. To overcome this issue, eight parallel accumulators are used. These latency values are stored in the n-bits sized registers.

Here, 8-bits input data, 11-bits registers are used. Data selector module takes an appropriate address of the data, which is present inside the data buffer. At each clock cycle, this data buffer makes available for four parallel data to the first stage adder. At the succeeding clock cycle, the data buffer provides an extra of four data to the adder stage 1 and the result of successive additions are applied to adder stage 2. Hence, the clock latency of stage 2 is two clock cycles. Multiplexer selects the data either from the column sum registers or from the adder stage 2 and the adder adds the consecutive samples. Only one register is used to store the binary values of cosine in fractional form. The values of the ACCi are multiplied using a multiplier. As the ACCi values are produced every two clock cycles, 4 multiplications are executed in eight clock cycles and the results are accumulated.

The column sum is computed in initial eight clock cycles. After the computation of the column sum, further computations are performed within 12 clock cycles, i.e., successive DCT coefficients are obtained after every 12 clock cycles.
5.6.1 1D-DCT Pipelined Architecture

The DCT architecture has 8 points as the DCT input and output. All data in the DCT architecture are accessed in consecutive manner. Each input and output process consumes eight clock cycles. Usually, 8 points 1D-DCT computation process requires 22 clock cycles. The input and output process visualization is shown in Figure 5.16. The DCT system computes all steps in a single clock cycle, hence, the computation of DCT is found to be faster.

Figure 5.15 DCT coefficients with the clock latency

Figure 5.16 Visualization of clock cycles for 8 points 1-DCT
5.6.2 Transpose Buffer in DCT

Transpose buffer is a static RAM, which is modeled with two set of data and address bus. It has input and output data and address buses, and the structural construction of transpose buffer is shown in the Figure 5.17. The output from the initial 1D-DCT is provided as the input to the transpose buffer.

The controller module creates the address in, out and Write Enable (WE). The input address is created in the normal sequence, such as (0,1,2,3,4,5,6, …, 63). The output is created in the transposed sequence, such as, (0,8,16,24,32,40,48,56,1,9,17,..,55, 63). The initialization of the output process after 64 inputs provides the time latency between the first input and output. The results of the transpose buffer are provided as input to the second 1D-DCT unit and the process is repeated in the same order.

![Figure 5.17 Transpose buffer](image-url)
5.6.3 2-D Discrete Cosine Transform (DCT)

The 2-D DCT can be computed in multiple ways. Just by multiplying the input vector with raw DCT coefficients, the 2-D DCT can be easily computed. Though this method is algorithm less and fast, it needs large logic utilization, such as, multiplier. The Discrete Cosine Transform comprises a set of basis vectors. These vectors are sampled cosine functions.

The 2-D DCT of a data matrix is represented in Equation 5.1.

\[ T = MXM^t \]  
(5.1)

where,

\( T \) denotes the data matrix, \( M \) represents the matrix of DCT coefficient and \( M^t \) is the transpose of \( M \). An expanded form of the above equation is described as follows

\[
\begin{bmatrix}
M_{00} & M_{01} & \ldots & M_{07} \\
M_{10} & M_{11} & \ldots & M_{17} \\
\vdots & \vdots & \ddots & \vdots \\
M_{70} & M_{71} & \ldots & M_{77}
\end{bmatrix}
\begin{bmatrix}
x_{00} & x_{01} & \ldots & x_{07} \\
x_{10} & x_{11} & \ldots & x_{17} \\
\vdots & \vdots & \ddots & \vdots \\
x_{70} & x_{71} & \ldots & x_{77}
\end{bmatrix}
\begin{bmatrix}
M_{00} & M_{10} & \ldots & M_{70} \\
M_{01} & M_{11} & \ldots & M_{71} \\
\vdots & \vdots & \ddots & \vdots \\
M_{07} & M_{17} & \ldots & M_{77}
\end{bmatrix}
\]  
(5.2)

where, for \( N \times N \) data matrix,

\[ M_{a,l} = \sqrt{\frac{2}{N}} \cos \left( \frac{(2a-1)(l-1)\pi}{2N} \right) \]  
(5.3)

For \( a=1, 2, \ldots, N, l=1, 2, 3, \ldots, N \), \( M_{a,l} = N^{-1/2} \) for \( l=1 \)

The row-column decomposition technique is used to deploy the 2D-DCT. The transpose buffer stores \( X^tM \) that is obtained by calculating the
1D-DCT of all the columns present in the input data matrix. Each row in $M^t X$ when applies the 1-D DCT, produces 2D-DCT.

5.7 DCT ARCHITECTURE IN FPGA

The DCT proposed architecture includes adders, multiplexers, delays and shifters. For DCT, the output obtained from the DWT is given as input, which has 8-bits. The input is given to the adder 1 and shifter, and this process is categorized into two phases. In the first phase, the streams in the adder are passed to the 8-bit multiplexer and Delay1. The selected data in the 8-bit multiplexer is send to the delay2. The data transformed to D1 are sent to the adder 2, after performing the addition operation the final data are passed to the output multiplexers. The output of the delay2 is given as feedback to the adder where the adder adds the bit streams and passes it to the multiplexer. This process takes place recursively. The output of delay2 is given to the adder 3 where the data is transmitted to two different delays, namely D3 and D4. In the D4, a delay is performed and finally the data are passed to the output multiplexer. The bit streams in the delay 3 are sent to the adder 5 and later to the output multiplexer. Figure 4.18 illustrates the system design for DWT in FPGA.

Figure 5.18 System design for DCT in FPGA
In the second phase, the input stream of DWT is given to the shifter, where the bits are shifter to the left side. Then, these bit streams are directed to the adder 4 to perform addition operation. The final output is send to the output multiplexer. The output multiplexer selects the data, which is lastly obtained as 8-bit DCT output.

\[
\begin{align*}
\text{DCT}_{\text{Coeff}} = & \begin{bmatrix}
49 & 50-8 & 51-8 & 52-24 & 53-32 & 54-40 & 55-48 & 56-56 \\
70 & 71-8 & 72-8 & 73-24 & 74-32 & 75-40 & 76-48 & 77-56 \\
77 & 78-8 & 79-8 & 80-24 & 81-32 & 82-40 & 83-48 & 84-56 \\
84 & 85-8 & 86-8 & 87-24 & 88-32 & 89-40 & 90-48 & 91-56 \\
98 & 99-8 & 100-8 & 101-24 & 102-32 & 103-40 & 104-48 & 105-56
\end{bmatrix}
\end{align*}
\] (5.5)

5.7.1 State Flow Diagram for DCT

The state diagram for the DCT transformation is given in Figure 5.19. The detailed explanation of the state flow diagram are described below.

ALGORITHM

The CLK and RESET option is enabled

STEP 1: The output of DWT is loaded.

STEP 2: Input bit streams are transformed to registers

STEP 3: Bit streams is forwarded to adder and multiplier to perform some operation

STEP 4: Coefficient of bit streams are updated

STEP 5: Checks for iteration limitation
If iteration attains the limitation, then process is transferred to the next level

STEP 6: Else

Look up table is checked

STEP 7: Convolution operation is performed

STEP 8: Weight of the bit streams are updated

STEP 9: Updated weights are accumulated

STEP 10: Process is transferred to the next stage.

Figure 5.19 State flow diagram for the proposed DCT system
5.7.2 Detailed Processing of DCT in FPGA

Input:

Clk, enable, start, Din

Output:

rdy, Dout.

Processing:

Din = ‘10100000’;

Address = 1;

iters = 1;

For each raising edge of clock pulse,

If enable = 1 and start = 1, then

Address = modulo (address + 1) by 64 = 1;

If (iters/63) = 1, then

iters = iters +1;

Shiftr = Din &shiftr = ‘01010000’;

DCTv = DCTcoeff(address) = ‘00110001’

Dout = Shiftr&DCTv = ‘1110001’

end;

5.7.3 Quantization in DCT

By dividing the entire 2D-DCT coefficient into quantizing values, the quantization process of the JPEG image compression is computed. The quantization table shown in Figure 5.19 provides the quantizing values. This
quantizer module consists of ROM and divider. These quantizing values are first stored in ROM. The divider carries out division in a pipelined manner. The first DCT coefficient coming out from 2D-DCT module is divided by the first value from the quantization table (which was already stored in ROM). The quantization matrix obtained is shown below.

\[
\begin{array}{cccccccc}
16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\
12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\
14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\
14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\
18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\
24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\
49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\
72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \\
\end{array}
\]

The second DCT coefficient is divided by second value from the table, like wise total 64 coefficients are divided by the values in quantization table. In this quantization process also we used pipeline architecture. A detailed description of our research is depicted as block diagram in Figure 5.20.

![Figure 5.20 Quantization and zigzag architecture](image)
5.7.4 Zigzag Buffer

In our research work, zigzag buffer is used. It is created using the static RAM. The creation of the zigzag buffer is same as the transpose buffer. The zigzag buffer has two datasets, such as, address bus and input address bus. Generally, input address bus is retrieved by normal sequence method, but output address is given to the zigzag patterns through zigzag sequence. Zigzag address is produced by a zigzag RAM and the sequence is warehoused in the RAM. The RAM data bus generates the Zigzag values, when it is accessed by the normal address sequence. The zigzag ordering for DCT coefficients is shown in Figure 5.21.

![Zigzag ordering for DCT coefficients](image)

Figure 5.21 Zigzag ordering for DCT coefficients

5.8 IMPLEMENTATION OF RUN LENGTH ENCODING IN FPGA

The 8-bit streams are given as input registers, which is forwarded to the comparator. The comparator compares the input bit streams and transfer to the count register. In the count register, the values are stored in different registers. It checks for the previous values stored in the count register, if the
value stored in the count register is same then the counter value is incremented to 1. Else, the counter value is assigned to 1. The RLE encoding rejects the repeated data, which results in compressed data. The compressed data and the value stored in the count register is shifted and multiplexed. The multiplexer chooses the significant data, which is obtained as final output. Figure 5.22 shows the system design for RLE in FPGA.

![System design for RLE in FPGA](image)

**Figure 5.22 System design for RLE in FPGA**

### 5.8.1 State Flow Chart Diagram

The steps involved in the RLE are shown in Figure 5.23 and a detailed description in explained in the algorithm.

In RLE, first the clock and reset feature is enabled.

- **STEP 1:** Input binary streams are loaded
- **STEP 2:** The binary streams are loaded into the registers
- **STEP 3:** If \( I_{in} = I_{out} \)
- **STEP 4:** Then perform \( count = count + 1 \)
STEP 5: if the input is not equal to output, the count is initialized to 1

STEP 6: The values are assigned to RLE count

STEP 7: If I_{out}=I_{inprev} RLE encoding is done

STEP 8: RLE encoding is given to STEP 3.

Figure 5.23 State flow diagram for the proposed RLE system

5.8.2 Detailed Processing of RLE in FPGA

Input:

Clk, rdy_in, Din.

Output:

Datacount, RLE_out, Output_En, eob.

Processing:

Output_En = 0;
Din = ‘1110001’;

Count = 1;

For each raising edge of Clk,

reg1 = Din;

reg2 = ‘00000000’;

If rdy_in = 1, then

If reg1 = reg2, then

Count = count+1;

reg2 = reg1;

Output_En = 1;

Else

Count = 1;

reg2 = reg1;

Output_En = 1;

If eob = 1

Output_En = 1;

Datacount = count;

RLE_out = reg2;

5.9 FLOORING DESIGNS OF DWT, DCT, AND RLE IN FPGA

The flooring process allocates the circuit components into a chip region. The flooring process by considering a finite set of cells/macros, chip outline and a net list allocates the predesigned cells/macros to estimated
positions on the chip. The allocation is done efficiently, such that, no two cells/macros overlie each other. The traditional flooring design problem pursues to reduce the wire length under the constraint that cells/macros do not overlie. The issues related to the huge complexity of the modern circuit design are predesigned macro blocks and minimal timing and routability optimization. The reuse of the predesigned macro blocks, such as, analog blocks, predesigned data paths, embedded memories generates multiple issues. Due to the design complexity, scaling of devices and interconnects, the process of optimizing the timing and routability becomes more difficult. The modern design challenges are incorporated to handle these placement problems. As multiple objects that has different sizes and large-scale designs are need to be processed, the modern placement problem becomes very difficult. In our proposed architecture, the number of components, area consumption, wire length and critical path delays are reduced when compared to the existing architecture system. The features involved in the proposed architecture design are described below,

- This system cooperates the high level of the design hierarchy and also the low-level elements, like, I/Os, function generators, tristate buffers, flip-flops, and RAM/ROM.
- Using the complex patterns, the logic structures like the interleaved buses are benefited.
- The system inevitably allocates the logic into columns or rows
- The rats nest connections are represented using the dynamic rubber banding
- By utilizing the name or the connectivity information, it discovers the logic or nets.
• By rearranging the hierarchy, the system design makes the floor planning process simple.

• The connectivity or function is based on the groups logic

• It delivers an alternative Unregistration Confirmation (UCF) flow that permits the user to formulate the graphically made limitations to a UCF

• It also permits the binding and unbinding of Relatively Placed Macro (RPMs)

• Recognizes the problems that are related to placement in the Floor plan window

![Figure 5.24 Flooring diagram of DWT, DCT and RLE in FPGA](image)

**5.10 ROUTING DESIGNS**

Routing describes the paths that are suitable for the conductors to carry their electrical signals on the chip layout. Using the conductor electric signals, the pin that has the same electrical signals is connected. Once the routing process is completed, the physical verification processes, such as, design rule checking, performance checking and reliability checking are
implemented. The physical verification process validates whether the circuit timing, geometric patterns and electrical effects satisfies the design rules and specifications. The process of routing is also an important problem that needs focus. The traditional routing system, named, To make it manageable addresses the routing problems using the two stage technique. The two stage technique includes global routing and detailed routing stages. Initially, the global routing splits the entire routing region into tiles/ channels, and then optimizes the specified objective functions using the tile-to-tile paths of all nets. (e.g., whole wire length and the critical timing constraints). Finally, based on the results of the global routing and design rules, the detailed routing calculates the actual tracks and routes for all the nets. In the Figure 5.25, gray color denotes the buses, which plays a significant role in the transmission of data. The pink color indicates the connection between two devices. The slices present in the design are represented in blue color, which is an elementary programmable logic blocks in FPGA. The CLB are represented in green color, which forms a switching matrix with some selection ciiultry.

Figure 5.25 Routing designs of DWT, DCT and RLE in FPGA
5.11 IMPLEMENTATION OF PRINCIPAL COMPONENT ANALYSIS (PCA) AND SINGULAR VALUE DECOMPOSITION (SVD) IN FPGA

The issues related to the time efficiency of the statistical methods like PCA and SVD are reduced by implementing them in FPGA architecture. The architecture for 8-point PCA is shown in Figure 5.26. It consists of 16 identical Processing Elements (PE’s). Each PE consists of a parallel Baugh-Woolley multiplier, storage elements where the coefficients $h_{ij}$ and $h_{ij}$ are stored in a storage element for pipelining the partial products. Based on the fast carry, the partial adder adds the result of the partial product with the previous result. FPGA-based PCA is tuned for hardware implementation and software realizations. Hik is a coefficient value and Xi and Yi are the coordination value of image pixels. The implementation cost of FPGA-PCA is low and also offers a high ratio of compressed images.

Figure 5.26 Architecture of PCA in FPGA for N=4

The inputs of PCA are Hik, Xi and Yi where Hik and Xi are given to the storage elements and Yi is given to full adder. The structure of the processing element is given in the Figure 5.27. The input data elements $i$ in the north direction are provided in a parallel manner. The matrix elements of the kernel
that are positioned in the corresponding PE cells for the entire calculation are also fed parallel. The kernel matrix elements are stored in the storage element and the final output is \( Y_{i\text{out}} \), which is obtained from the storage element.

![Figure 5.27 Structure of processing element (PE)](image)

The complex valued samples resulting obtained from input bins are stored into a FIFO with independent clocks. Once appropriate samples are stored in the FIFO, the load logic builds the matrix format desired into the four dual ports RAMs, one RAM per element of a 2x2 PE. This allows the contents of an entire PE to be retrieved or stored in a single clock cycle. Once the master state machine identifies that the matrix is full, the elements of the diagonal PEs are streamed through the fully parallel first and second transformations, storing the significant rotation parameters into many smaller dual port RAMs.

![Figure 5.28 Architecture diagram of SVD](image)
Swapping process is carried out by swap logic, after storing the rotation parameters, the entire contents of loaded RAM are fed through a parallel multiple 2x2 multiplication. The results are streamed into an output registers, from which the elements can be swapped at a higher clock rate. The architecture of SVD is shown in Figure 5.28 and the final output is obtained from output registers.

5.12 RESULTS OF PROPOSED FPGA BASED DWT-DCT-JPEG AND DWT-DCT-PCA TECHNIQUES

Table 5.1 provides the device utilization summary of the proposed method. The measured data from the toolkit is routed back the MATLAB simulation for HSIs reconstruction. The architectures are designed using Xilinx system generator tools within the Matlab software framework. Table 5.2 shows the parameters used for FPGA based DWT-DCT-JPEG technique.

Table 5.1 Device utilization summary of DWT-DCT-JPEG

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slice Registers</td>
<td>170</td>
<td>18,224</td>
<td>1%</td>
</tr>
<tr>
<td>Number of Slice LUTs</td>
<td>178</td>
<td>9,112</td>
<td>1%</td>
</tr>
<tr>
<td>Number of occupied Slices</td>
<td>72</td>
<td>2,278</td>
<td>3%</td>
</tr>
<tr>
<td>Number of MUXCYs used</td>
<td>8</td>
<td>4,556</td>
<td>1%</td>
</tr>
<tr>
<td>Number of LUT Flip Flop pairs used</td>
<td>231</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Number of fully used LUT-FF pairs</td>
<td>97</td>
<td>231</td>
<td>41%</td>
</tr>
<tr>
<td>Number of bonded IOBs</td>
<td>89</td>
<td>186</td>
<td>47%</td>
</tr>
</tbody>
</table>
Table 5.2 Implementation result

<table>
<thead>
<tr>
<th>Parameter</th>
<th>FPGA based DWT-DCT-JPEG (8bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMOS Technology</td>
<td>45nm</td>
</tr>
<tr>
<td>Supply Voltage (Vdd)</td>
<td>1.1 V</td>
</tr>
<tr>
<td>LUT</td>
<td>178</td>
</tr>
<tr>
<td>FF</td>
<td>170</td>
</tr>
<tr>
<td>IOBs</td>
<td>89</td>
</tr>
<tr>
<td>Logical Elements</td>
<td>142</td>
</tr>
<tr>
<td>Frequency</td>
<td>157.359MHz</td>
</tr>
<tr>
<td>Area (mm$^2$)</td>
<td>0.232 mm$^2$</td>
</tr>
<tr>
<td>Total Power (mW)</td>
<td>20.40</td>
</tr>
<tr>
<td>Dynamic Power (mW)</td>
<td>0.48</td>
</tr>
<tr>
<td>Quiescent Power (mW)</td>
<td>19.92</td>
</tr>
</tbody>
</table>

Figure 5.29 Timing diagram for the proposed FPGA based DWT-DCT-JPEG
Figure 5.29 shows the timing diagram for the proposed FPGA based DWT-DCT-JPEG. The first row in the timing diagram denotes the clock pulse, which is represented as clk2.Enable is set to enable the total system, level shows the state level for the proposed algorithms such as state level for dwt, dwt_w, dwt_i, dwt_l, dwt_sd, swt_sde, and swt_sd_c. data count represents the count for the data and append 1 for the repeated data. The value tab shows the corresponding output for the final clock pulse, it is highlighted as yellow color in the timing diagram.

**Figure 5.30** Timing diagram for the proposed FPGA based DWT-SVD-PCA
Figure 5.30 illustrates the timing diagram for the proposed FPGA based DWT-SVD-PCA. The representation of \textit{clk2, Enable, level, data count,} and \textit{value} tab is similar to the Figure 5.30. Especially, the \textit{clk, reset, x_a, x_b} are given as an input to the proposed FPGA based DWT-DCT-PCA, and the output is ready, \textit{final_a, final_b}. Finally, the remaining intermediate results are signals. Table 5.3 illustrates the summary of the device utilization for FPGA based DWT-DCT-PCA technique.

**Table 5.3 Device utilization summary of the FPGA based DWT-SVD-PCA method**

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slice Registers</td>
<td>112</td>
<td>4,800</td>
<td>2%</td>
</tr>
<tr>
<td>Number of Slice LUTs</td>
<td>174</td>
<td>2,400</td>
<td>7%</td>
</tr>
<tr>
<td>Number of bonded IOBs</td>
<td>65</td>
<td>102</td>
<td>63%</td>
</tr>
<tr>
<td>Number of occupied Slices</td>
<td>56</td>
<td>600</td>
<td>9%</td>
</tr>
<tr>
<td>Number of Slice LUTs</td>
<td>174</td>
<td>2,400</td>
<td>7%</td>
</tr>
<tr>
<td>Number of LUT Flip Flop pairs used</td>
<td>177</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

5.13 SUMMARY

Some of the advantages of FPGA in image compression are described below,

- Fast prototyping and turnaround – Prototyping can be explained as constructing an actual circuit to a theoretical design to validate that it works, and to offer a physical platform for debugging it. Turn around refers the total time between submission of a process and its completion. Since in FPGAs all interconnects are already present and the
designer only has to fuse these programmable interconnects to get the desired logic output, the time taken is quite less.

- **NRE cost is zero**- Non-recurring engineering (NRE) denotes that the cost for the research, design, develop and testing phases of the new product are computed only once. Since the FPGAs are reprogrammable and can be utilized without any loss of quality every time, the nonrecurring cost is not present. This highly decreases the initial cost of manufacturing of ICs since the programs can be run and tested on the FPGAs free of cost.

- **High speed**-Since the FPGA technology is based on look-up tables, the time taken to execute is less than that in ASIC technology. This high speed is used in making various multipliers today.

- **Parallel processing**- FPGAs discover applications from multiple area and algorithms that exploits their parallelism. Code breaking areas, such as, brute-force attack and cryptographic algorithms are popularly used by the FPGA. Even at the minimal MHz clock rates, the logic resources on the FPGA consumes less computational throughput. Using number format for trading off precision and parallel range of arithmetic units, the performance of the FPGA is greatly increased.

- **Low cost**-The cost of FPGA is quite affordable and hence it makes them very designer friendly. Also the power requirement is less since the architecture of FPGAs is based on LUTs.
Due to the above mentioned advantages of FPGAs, it is used for image compression. In our research, the Jpeg hyper spectral image compression is designed in VHDL and implemented in FPGA. The reduction of FPGA area and power is done by using DWT with DCT combination. The RLE coding architecture has been implemented with the reduced memory and it encodes the coefficients bit-by-bit at each clock cycle resulting in an efficient design. The use of FPGA in image compression reduces the area and power consumption because in the proposed FPGA technique utilizes less number of flip flops and LUTs. Though, it consumes less area and power, time taken to compress an image and computation cost is high. This is reduced by implementing the SVD and PCA in FPGA, which utilizes less storage space, has simple structure of bit rate stream. This scheme also provides a large degree of flexibility in compressing the hyperspectral images.