2D-DCT and Quantization Accelerator for video codecs on MPSoC FPGA using OpenCL framework and Neutrosophy

Sumalatha S.[1,*], Rajeswari[2]

[1]Research Scholar, Acharya Institute of Technology, Visvesvaraya Technological University, India

[2]Professor and Head, Acharya Institute of Technology, Visvesvaraya Technological University, India

Emails: sumalatha.disha@gmail.com; rajeswari@acharya.ac.in

Abstract

Video codecs based on lossy compression techniques take advantage of removing redundant data in spatial and frequency domains. The various modes of intra- and inter-predictions help to reduce the redundant information in the spatial domain in standard video codecs like AVC, HEVC, and VVC. Further, the removal of redundant information in the frequency domain is achieved by adaptive quantization of transformed frames obtained after DCT-II or DST transformation techniques. In traditional video codec standards, adaptive quantization matrices are derived using the Human Visual System (HVS) model and display resolution parameters, which adjust the quantization step size to preserve perceptually significant pixel information in transformed blocks. The Neutrosophic (NS)-based approach introduces a more refined mechanism for generating the quantization matrix, utilizing Neutrosophic set membership values (true, indeterminate, and false) assigned to each region or frequency component of the transformed block. These values reflect the certainty of pixel relevance, enabling a more adaptive, perceptually driven quantization process. The proposed method incorporates NS logic in combination with the Human Visual System (HVS) model and display resolution parameters. By blending these factors, the quantization step size is optimally tuned to enhance visual quality. The HLS implementation of the transformation and quantization technique suitable for video codec acceleration using the OpenCL framework is adopted in our work. The design was implemented and tested on the Xilinx ZCU-104 board using a standard test sequence from the JCTVC and UVG datasets of various resolutions and diversified content. The testing showed an optimized resource utilization of 60.36%, with notable metrics indicating perceptually good results. The objective metrics showed an improvement of 3.77% in PSNR and 1.83% in SSIM compared to standard HVS-based quantization.

Keywords: Discrete Cosine Transformation; Human Visual System; Vitis HLS; Neutrosopic matrices; Adaptive Quantization; Structural Similarity Index Metric

1. Introduction

At present, a significant portion of global internet traffic is attributed to the streaming of Full HD and Ultra-HD videos, driven by the widespread availability of smart consumer electronics. However, communication channels and storage media inherently possess finite capacities, necessitating more efficient video compression techniques that can surpass the bitrates of High Efficiency Video Coding (HEVC) [1] without compromising visual quality. Existing video coding standards are continuously evolving, integrating advanced features and optimized methods to compress HD and UHD video frames, particularly for consumer applications. Despite these advancements, the core encoding and decoding pipeline remains based on the hybrid coding model, which includes block-based partitioning followed by transformation and quantization of the residuals. The latest Versatile Video Coding (VVC) standard [2] introduces more flexible block partitioning structures, such as the Asymmetric Quad Tree with Multi-Type Tree (QTMT), enhancing adaptability in spatial decomposition. These blocks are subsequently coded using spatial and temporal prediction techniques.

DOI: https://doi.org/10.54216/IJNS.270222

Neutrosophic logic has shown promise in various image and video processing tasks, including classification, segmentation, denoising, medical imaging [3,4,5], encoding [6] and compression [7]. Neutrosophic logic can be applied to both spatial and frequency domain video compression techniques to improve performance by handling uncertainty and imprecision inherent in video frames. This approach involves representing frames using neutrosophic sets, which incorporate degrees of truth, indeterminacy, and falsity for each pixel. Usually, the frequency domain video compression involves the frame transformation from spatial domain into frequency domain using DCT/DST transformation methods. Then Neutrosophic logic can be integrated into the quantization process [7], where high-frequency components often associated with noise and less visible details can be more aggressively reduced or eliminated based on their neutrosophic characteristics such as high indeterminacy or falsity. Neutrosophic logic can effectively handle noise and uncertainty in frames, leading to better compression results without significant quality loss. Neutrosophic logic allows for adaptive compression, where the degree of compression can be adjusted based on the specific characteristics of different frame regions or position in the transformed block.

In earlier standards like AVC and HEVC, the Discrete Cosine Transform Type-II (DCT-II) was the primary transform used. VVC extends this by incorporating DCT-II, DST-VII, and DCT-VIII transforms into more efficient residual coding [8]. Since these transforms are mathematically defined over real numbers, hardware implementations necessitate their integer approximations to enable efficient computation. Researchers have proposed integer transformation kernels for DCT-II that preserve orthogonality and basis vector properties, allowing energy compaction in low-frequency regions [9]. The AVC standard employs a multiplier-less 4×4 fixed-point DCT implementation with 16-bit precision [10], while HEVC utilizes advanced schemes such as constant matrix multiplication (CMM), multiple constant multiplication (MCM) [10], butterfly structures [11], and shift-add approximations [12] to support larger block sizes like 8, 16×16, and 32×32 with integer approximated coefficients. In VVC [13], DST-VII transforms are approximated using integer DCT-II kernels with additional adjustment stages, formulated as an optimization problem to balance orthogonality and approximation error while satisfying sparsity constraints. DCT-VIII transforms are derived through post-processing of the DST-VII approximation using permutation and sign modification techniques. Following transformation, quantization is performed using adaptive quantization matrices tailored for human visual system (HVS) sensitivity and the target display resolution [14,15].

The development of hardware-accelerated video codec components using integer transforms and adaptive quantization matrices must consider algorithms which are hardware compatible, require short development cycles, and minimal hardware expertise. To meet these demands, several researchers have proposed leveraging OpenCL, HLS, and MATLAB Simulink for rapid prototyping and deployment of transformation and quantization IP cores on heterogeneous platforms. The research work [16] demonstrates an OpenCL-based JPEG compression pipeline targeting GPGPU and multi-core CPUs, achieving speedups of 7.97× and 8.65× for 1024×1024 and 2048×2048 images, respectively. In [17], the intra-frame transforms and quantization pipeline for H.266/VVC was implemented using the Intel FPGA SDK and deployed on a Stratix 10 FPGA, achieving throughput ranging from 6.5 fps to 83.3 fps for 2K resolution depending on coding unit size. Another HLS-based design targeting inverse transform and dequantization [18] achieved 13 fps for 2K video on the ZC-702 SoC platform, while demonstrating 60% power savings over pure software implementations. Ben Atitallah et al. [19] compared high-level HLS and low-level RTL implementations of HEVC inverse quantization and inverse transformation on a Xilinx XC7Z045 FPGA. The HLS design achieved 2K@24 fps at 175 MHz, whereas the RTL implementation achieved 4K@26 fps at a lower frequency of 145 MHz, highlighting the performance trade-off between design abstraction levels.

2. Motivation and Contribution

The high-level synthesis (HLS) implementation of transformation and quantization kernels on heterogeneous edge computing platforms, particularly reconfigurable FPGAs coupled with Application Processing Units (APUs) or Real-Time Processing Units (RPUs) has significant potential in consumer electronics for video compression applications [20]. Key design considerations for accelerator development on such edge devices include resource constraints, processing speed, and power efficiency. This work explores the use of the Xilinx Vitis Vision [21] Library to develop an accelerator kernel for a unified video encoder-decoder, employing integer-approximated transformation and quantization operations optimized for FPGA [22] deployment. A Neutrosophic logic-based approach is used to generate an efficient quantization matrix that preserves pixels with high truth membership while allowing more aggressive compression of pixels with high indeterminacy, thereby reducing storage requirements with good perceptual results. The accelerator kernel is further optimized using techniques such as pipelining, loop unrolling, memory interface mapping to separate memory banks for parallel data transfers across multiple computing units. The proposed design is evaluated using standard video datasets, including JCTVC [23] and UVG [24], which offer diverse content and resolution profiles. Hardware implementation results covering resource utilization, throughput, and objective quality metrics such as PSNR and SSIM are presented, along with comparative analysis against existing state-of-the-art approaches discussed in the literature.

DOI: https://doi.org/10.54216/IJNS.270222

3. Methodology

This section presents the complete mathematical framework used to obtain integer kernel coefficients for the transformation operation, along with the application of Neutrosophic logic for adaptive quantization matrix derivation. Further, the proposed architectural framework developed using OpenCL programming model is described in detail. OpenCL uses Xilinx Vitis unified software platform, which offers significant advantages for data acceleration by providing a high-level programming framework that facilitates the design, and deployment of accelerator functions on heterogeneous hardware architectures. Specifically, it supports systems composed of an ARM-based Processing System (PS) and an Accelerator Kernel implemented on Programmable Logic (PL) connected via an AXI interface. The Xilinx Runtime (XRT) enables OpenCL-based applications running on a Linux environment to orchestrate the execution of custom accelerator kernels such as transformation and quantization in the encoder path, and dequantization and inverse transformation in the decoder path of a video codec, developed as an IP on the FPGA fabric. Data transfers among the ARM processor, the FPGA, and the on-chip external DDR memory are also handled by XRT. The subsequent section details the algorithmic implementation of the accelerator kernel targeted for execution on the PL, as well as the host-side control logic running on the PS, which manages data movement and kernel execution using the OpenCL programming model.

In the proposed design, video frames are first converted from spatial to frequency domain using standard DCT-II transformation, which is the primary transformation technique used in all video codec standards. The 2D – DCT equation is given by

$$T_{x,y} = \begin{cases} \frac{1}{\sqrt{N}} & \text{if } x = 0 \\ \sqrt{\frac{2}{N}} \cos \frac{(2y+1)x\pi}{2N} & \text{if } x > 0 \end{cases} \quad (1)$$

Where $T_{x,y}$ represents the element value of the transform coefficient at position x,y.

x is the row index, y is the column index, N is the transform size and x,y = 0,1. ., N − 1.

The transform coefficients in equation (1) are real numbers, which are not computationally flexible for hardware implementation like FPGA. Hence, the transform coefficients of eqn (1) are approximated to integers using eqn (2). This approximation is required for accelerating the DCT process in both forward and inverse transformation operations.

$$T_{x,y} = \text{round}\left[2^p * t_{x,y}\right] \quad (2)$$

Where $p=6+\frac{log_2^N}{2}$ and N= transformation size.

A similar approach is used to derive a quantization matrix based on the human visual system (HVS), display resolution and the positional importance of transformed pixel position within the transform block based on the Euclidean distance from DC coefficient to other AC coefficients in the block, which is modified using novel Neutrosophic concept.

Let us consider the relationship between the standard quantization matrix and the proposed adaptive quantization matrix, which depends on the above-mentioned parameters, defined in eqn (3)

$$H'_{i,j} = H_{i,j}^{Mi,j} \quad (3)$$

Where Hi,j is the standard quantization matrix based on HVS given in [14] based on the perceptual importance of DCT values in the transformed domain.

Mi, j is the Modification parameters relied on display resolution and the low frequency ac coefficients of the transformed pixel in the Transform Unit (TU).

Hi,j' is the adaptive quantization matrix based on HVS, display resolution and modified Euclidian distance matrix using Neutrosophic logic.

The modification parameter $M_{i,j}$ is given in eqn (4).

$$M_{i,j}\left(Eud_{i,j_{NS}}, W\right) = e^{-\left[\frac{Eud_{i,j_{NS}}}{W}\right]} \in [0,1] \quad (4)$$

Where Eud_{i,j_NS} is the Euclidean distance of the current ac pixel from dc in the transform block, modified using Neutrosophic (NS) technique.

DOI: https://doi.org/10.54216/IJNS.270222

W is the display resolution parameter based on a normalized hypotenuse value.

In the next section, discussed the detailed derivation of Neutrosophic based Euclidean distance matrix from the standard Euclidean distance measurement at the given pixel position, given by

$$d_{(i,j)} = \sqrt{\frac{(i_1-i_2)^2+(j_1-j_2)^2}{(i_1-i_{max})^2+(j_1-j_{max})^2}} \in [0,1] \qquad (5)$$

Where i_1, j_1 is the pixel of DC position i.e., at (0,0) in the block.

i_2, j_2 are the pixels at AC position in the block.

i_{max}, j_{max} are the pixels at the last AC position in the block.

Now for a block size of 8x8 the obtained Euclidean distance matrix from eqn (5) is given below

d(i,j)=

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0.0000	0.1010	0.2020	0.3030	0.4041	0.5051	0.6061	0.7071
1	0.1010	0.1429	0.2259	0.3194	0.4165	0.5151	0.6145	0.7143
2	0.2020	0.2259	0.2857	0.3642	0.4518	0.5440	0.6389	0.7354
3	0.3030	0.3194	0.3642	0.4286	0.5051	0.5890	0.6776	0.7693
4	0.4041	0.4165	0.4518	0.5051	0.5714	0.6468	0.7284	0.8144
5	0.5051	0.5151	0.5440	0.5890	0.6468	0.7143	0.7890	0.8690
6	0.6061	0.6145	0.6389	0.6776	0.7284	0.7890	0.8571	0.9313
7	0.7071	0.7143	0.7354	0.7693	0.8144	0.8690	0.9313	1.0000

Now based on Neutrosophic concepts [3,4], the above Euclidean distance matrix values are modified into 3 regions of values that defines the quantization levels of each transformed pixels in the transformation block as follows

1) The truth-values (T) of blocks representing high-importance regions, which will have lower quantization steps to maintain higher precision of the transformed value.

2) The indeterminacy values (I) indicate regions of uncertainty, such as noisy or unclear areas. These regions can tolerate moderate quantization steps, as they are less important for visual quality.

3) The falsity values (F) represent areas of the video that are irrelevant or unnecessary. These regions can be compressed with the highest quantization steps, leading to a greater reduction in data without affecting the perceived quality.

To obtain final modified Euclidean distance matrix which decides the importance of transformation pixels in transform block from the above definitions and the application of Neutrosophic logic as a combination of {T (i , j), I (i , j), F (i , j)} matrices Where T=truth matrix, I=Indeterminant matrix, F=false matrix to d(i,j), is described in detail in the following steps.

Step-1: construction of truth value matrix T (i,j) given as

$$T(i,j) = \frac{\bar{g}(i,j)-\bar{g}_{min}}{\bar{g}_{max}-\bar{g}_{min}} \qquad (6)$$

Where $\bar{g}(i,j)$ is the mean pixel value of the Euclidean distance matrix d(i,j) by padding the edge value of the matrix by the number of rows and columns of size equal to 'p1'. The value of 'p1' is decided from the size of the block considered for mean calculation.

$\bar{g}_{max}$ and $\bar{g}_{min}$ are maximum and minimum value in $\bar{g}(i,j)$ matrix.

The mean value of the padded d(i,j) matrix can be derived using eqn (7)

DOI: https://doi.org/10.54216/IJNS.270222

$$\overline{g}(i,j) = \frac{1}{p1 \times p1} \sum_{m=i-\frac{p1}{2}}^{m=i+\frac{p1}{2}} \sum_{n=j-\frac{p1}{2}}^{n=j+\frac{p1}{2}} d(m,n) \quad (7)$$

Compute the maximum and minimum values of the local mean value matrix $\overline{g}(i,j)$ by eqn (8)

$$\overline{g}_{max} = \max\left(\overline{g}(i,j)\right) \quad \overline{g}_{min} = \min\left(\overline{g}(i,j)\right) \quad (8)$$

Step-2: Construction of indeterminate matrix I (i,j) given in eqn (9) by computing the difference value between the d(i,j) and its local mean intensity given in eqn (10), and the maximum and minimum value of the divergence matrix using eqn (11).

$$I(i,j) = \frac{\delta(i,j)}{\delta_{max} - \delta_{min}} \quad (9)$$

$$\delta(i,j) = \left|\left(d(i,j) - \overline{g}(i,j)\right)\right| \quad (10)$$

$$\delta_{max} = \max\left(\delta(i,j)\right) \quad \delta_{min} = \min\left(\delta(i,j)\right) \quad (11)$$

Step-3: Construct the false matrix F (i,j), computed as difference value matrix between the original Euclidean distance matrix and its mean intensity matrix, which is defined in eqn (10)

$$F(i,j) = \delta(i,j) \quad (12)$$

Step-4: Aggregate all three matrices to get a new Euclidean distance matrix based on the weight's assignment to T, F and I matrices as per the definitions considered from the Neutrosophic technique in the proposed work. The final new Euclidean distance matrix is given in eqn (13)

$$Eud_{i,j_{NS}} = \frac{0.9 \times T(i,j) + 0.2 \times F(i,j) - 3 \times I(i,j)}{3} \quad (13)$$

The modification parameter in eqn (4) for calculating the new quantization matrix also depends on the display resolution of the targeted application. Display resolution parameter 'w' is computed using eqn (14). As per standards, the theoretical maximum value of screen width and height is 65536x65536. The normalized hypotenuse value 'h' is the ratio of the hypotenuse value of the actual HD screen and the theoretical value as per the JPEG standard given in (15). For h_{actual} calculation, the actual row and column values of 2K HD resolution will be 1920x1080 and for 4K UHD resolution is 3840x2160.

$$w = h^{h}_{theoretical} \in [0,1] \quad (14)$$

$$h = \frac{h_{actual}}{h_{theoretical}} \in [0,1] \quad (15)$$

Where $h_{theoretical}$ is the theoretical hypotenuse value of display resolution.

h_{actual} is the actual hypotenuse value of display resolution.

The final integer quantization matrix (16) suitable for hardware acceleration is obtained by the quantization parameter and from eqn (3).

$$QM_{intra} = round\left(\frac{QP}{H'_{i,j}}\right) \quad (16)$$

Where Quantization parameter (QP) controls the quantization step size in the quantization matrix. The usually considered set of QP in all the video codec standards for quantization testing is 22,27,32 and 37.

The next section describes the proposed architectural framework and accelerator kernel algorithm for 2D-transformation and quantization operations. The Algorithm gives details of various optimization techniques used in high-level kernel implementation.

A. Proposed Architectural Framework and Accelerator Algorithm description

The OpenCL framework supports load-store architecture and hence the algorithm development should follow write, compute and read pipeline on the targeted heterogeneous computing hardware, which has the embedded ARM-A53 processor as the Processing system (PS) and FPGA fabric as Programmable Logic (PL) shown in Fig 1. In Vitis kernel flow, usually the host and Accelerator kernel occupy separate compute space, which is managed by the Xilinx Runtime (XRT) unit. The host code written with Open CL API calls would be taking care of data writing and reading operation from host memory to global memory and then to the buffers on the device like LUTRAM/URAM/BRAM. The XRT will control the accelerator kernel operation by reading and writing control register maps through s_axilite interface. Through this interface, it communicates all the required control and scalar input values like block size, frame size, QP to the kernel and manages the kernel execution on the PL device. Once the device completes the computation, the computed results will be read from device memory back to host memory.

DOI: https://doi.org/10.54216/IJNS.270222

Host code written will be compiled to run on an ARM processor using the GNU Arm cross-compiler, which creates an extensible and linking file (.elf). Similarly, the accelerator function code written using high level C language will be targeted to run on an FPGA device and will be compiled using vitis HLS. Then, AXI Interconnect is automatically instantiated by V++ linker to connect between ARM AXI Master Interfaces and FPGA slave interfaces of acceleration kernels. The high-level C/C++ coding of PL kernels has specific requirements like supported interfaces, clocks, reset signals, including optimization techniques like data-flow modelling, pipelining, and task level parallelism (TLP) which are included in the proposed design.

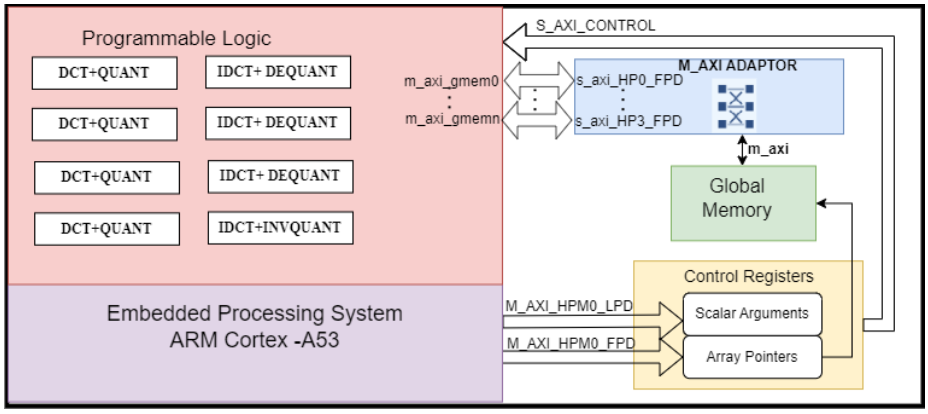

Figure 1. Proposed Architectural framework of High-level design.

In the proposed hardware-accelerated video codec design, two primary kernels are developed to handle the core transformations: one for Discrete Cosine Transform and Quantization (DCT+QUANT) in the forward processing path, and another for Inverse Discrete Cosine Transform and Dequantization (IDCT+DEQUANT) in the reverse path. Each of these kernels is instantiated as four Compute Unit (CU) instances, resulting in four parallel CUs in both the forward and reverse paths, as depicted in Fig. 1. The host application, executed on the ARM processor, prepares and transfers the required parameters to the FPGA device. These include the frame data divided into four slices (one for each CU), the row and column dimensions of the input frame, the Transform Coefficient Matrix (TCM) based on the selected TU block size, and quantization-specific parameters such as the type of quantization either Modified, based on Neutrosophic logic, or Regular, based on Human Visual System characteristics and the QP based Quantization Coefficient Matrix (QCM). Once the host transfers this information from host memory to device memory, it initiates the execution of the appropriate kernels.

To facilitate efficient memory access, data pointers are created for global memory banks using a memory-mapped AXI4 interface, allowing the input arrays to be loaded into the local buffers of the device. In addition to the primary array data, scalar arguments such as pointers to input and output buffers, and scaling factors applied after each transformation stage to maintain consistent data ranges are passed as control signals using the s_axi_control interface from the Processing System (PS) to the Programmable Logic (PL). To minimize memory access stalls during burst transfers of frame data, the input and output arrays are allocated to separate global memory banks, thereby maximizing memory bandwidth utilization and improving overall throughput.

Once all required data is available in the local memory of the four-compute units, the 1D row-wise transformation begins in parallel across the CUs. This transformation is implemented as a pipelined process with an Initiation Interval (II) of one, enabling the transformation of one TU block row per clock cycle. Further, Loop unrolling is employed to achieve full parallelism across the rows. The row-transformed data is then scaled and stored in the local buffers of the CUs. This data is subsequently transposed and passed through a column-wise transformation stage, which utilizes similar pipelining and unrolling optimizations to maximize efficiency. After completing the 2D transformation operation, quantization is applied to the transformed block in a pipelined manner, based on the quantization type and QP values received from the host.

Following the quantization operation, the output of the forward path (DCT+QUANT) is fed into the reverse path CUs (IDCT+DEQUANT), where dequantization and inverse transformation operations are performed. The final output, which represents the reconstructed frame data, resides in the FPGA's local memory and is subsequently transferred back to global memory. The next section outlines the algorithmic flow of the accelerated kernel design, highlighting the key optimization techniques employed to achieve high computational throughput and efficient hardware utilization.

DOI: https://doi.org/10.54216/IJNS.270222

Transformation and Quantization Kernel Algorithm

Initialization

int32_t rows = frame.rows;

int32_t cols = frame.cols;

int halfRows = rows / 2; // Halve the rows

int halfCols = cols / 2; // Halve the columns

Step1: Divide the input frame into four quadrants of equal size.

- Initialize an array quadrant of size 4 to store the quadrants.
- **For** i from 0 **to** 1 **do**:
- **For** j from 0 **to** 1 **do**:
  - Compute quadrantIndex as i * 2 + j.
  - Define the region of interest (ROI) roi using cv::Rect:
- roi.x is j * halfCols (start column of the ROI).
- roi.y is i * halfRows (start row of the ROI).
- roi.width is halfCols (width of the ROI).
- roi.height is halfRows (height of the ROI).
  - Extract the quadrant image using image (roi).clone() and store it in quadrants[quadrantIndex].
- **Output:**
- Return quadrants, which now contain the four quadrants of the frame.

Step 2: Pass all the control and data values to device memory used by Kernel for processing

//Control Interface definitions

s_axilite port=a_row, a_col, block_size, QP, TCM, QCM, SF1, SF2, SF3, SF4, return.

// data interface definitions

m_axi port ‘a’ mapped to gmem0, m_axi port ‘b’ mapped to gmem1, m_axi port ‘c’ mapped to gmem2,

m_axi port ‘d’ mapped to gmem3, m_axi port ‘e’ mapped to gmem4

#pragma HLS DATAFLOW

Step 3: Perform DCT/IDCT using Systolic Array and scale the intermediate results

//Initialize matrix C to zero.

For each element in C from 0 to block*block -1: c[i][j]=0;

// DCT operation

Outer Loop: for (int i = 0; i < block; i++) {

Middle Loop: for (int k = 0; k < block; k++) {

#pragma HLS PIPELINE II=1

int32_t f_val = (i < ROWS && k < COLS) ? localB[i][k] : 0;

Inner Loop: for (int j = 0; j < img_width; j++) {

#pragma HLS UNROLL

DOI: https://doi.org/10.54216/IJNS.270222

localC[i][j] += f_val * localA[k][j];}}}

Step 4: Scale the result after every multiplication to maintain data precision

```
for(int i=0;i<block;i++)
  for(int j=0;j<img_width;j++)
        localC[i][j] /= SF2;
```

Step 5: Quantization / De-quantization Kernel operations.

During forward path operation Quantization will take place otherwise dequantization in reverse path.

While blockRow < ROWS **do**:

- Initialize blockCol to 0.
- **While** blockCol < COLS **do**:
  - **If** quant_call mod 2 equals 1 **then**:

    //Perform Quantization by element-wise division operation.
  - Result_quant= DCT_output/Quantization matrix.
  - **Else**:

    //Perform inverse Quantization by element-wise multiplication operation.
  - Result_invquant= Quant_output *Quantization matrix.
  - Increment blockCol by 1.
- Increment blockRow by 1.

Return result.

Step 6: Send computed results

Send the processed result back to global memory from the device memory.

The three important optimization techniques in the proposed kernel design are

1. **#pragma HLS DATAFLOW:** This optimization technique helps task-level pipelined execution of the kernel. DATAFLOW directive will inform the Vitis High-Level Synthesis (HLS) compiler to run transformation and quantization operations concurrently, creating a pipeline of simultaneously running tasks.

2. **#pragma HLS PIPELINE II=1:** The pipeline directive pipelines the middle loop, in row and column transformation operations in both forward and inverse path, allowing new iterations to start at every clock cycle. This increases throughput by overlapping the execution of different iterations.

3. **#pragma HLS UNROLL:** The directive in the inner loop allows the compiler to execute all iterations of this loop in parallel. This reduces the time complexity of the inner loop to constant time, as all multiplications and additions for a given i and k are performed simultaneously.

4. Results and Discussions

This experimental study implements an OpenCL-based methodology to accelerate critical components of a video codec on a heterogeneous computing platform comprising an ARM processor and an FPGA, utilizing the Xilinx Vitis Unified Software Platform [21]. The host application, written in C/C++ with OpenCL API integration, is cross compiled using the GNU toolchain to generate an ELF executable. The programmable logic (PL) kernels, developed in a high-level synthesis (HLS) language and optimized via HLS directives, are synthesized to target the FPGA fabric [22]. Upon successful compilation and linkage, the host and kernel binaries are packaged alongside the platform-specific files, including the sysroot, root filesystem (rootfs), and the FPGA device image. The generated SD card image is used to boot the Xilinx ZCU104 evaluation board, which loads all necessary kernel modules, device tree configurations, and user applications, thereby enabling end-to-end execution of the accelerated video codec pipeline. The final hardware obtained after complete implementation includes four parallel

DOI: https://doi.org/10.54216/IJNS.270222

compute units implementing Discrete Cosine Transform (DCT) and Quantization, and four compute units for Inverse DCT (IDCT) and Dequantization, obtained as an HLS IPs depicted in Fig. 2

The proposed design is evaluated using standard test sequences from the JCT-VC dataset [23] and the Ultra Video Group (UVG) dataset [24]. Five video sequences are selected from the Common Test Conditions (CTC) defined by JCT-VC [23], covering a diverse range of content classes (Class B through Class F), along with one high-resolution sequence from the UVG dataset, representative of 4K Ultra HD content. These sequences are chosen due to their distinct characteristics tailored for specific video application domains. Class E sequences are primarily utilized to evaluate low latency use cases such as video conferencing. Class B and the UVG 4K sequence are employed to assess compression performance in high-definition (HDTV) and ultra-high-definition (UHD) multimedia scenarios. Class C and D sequences are selected to represent typical content encountered in mobile and portable device applications. Additionally, Class F includes screen content and computer-generated graphics captured via screen recording tools, making it suitable for evaluating scenarios involving text, GUI, and artificial visuals. Each class exhibits a unique combination of spatial and temporal complexities, including static and dynamic backgrounds, foreground motion, and varying degrees of object density, thereby providing a comprehensive validation framework for the proposed video codec acceleration architecture.

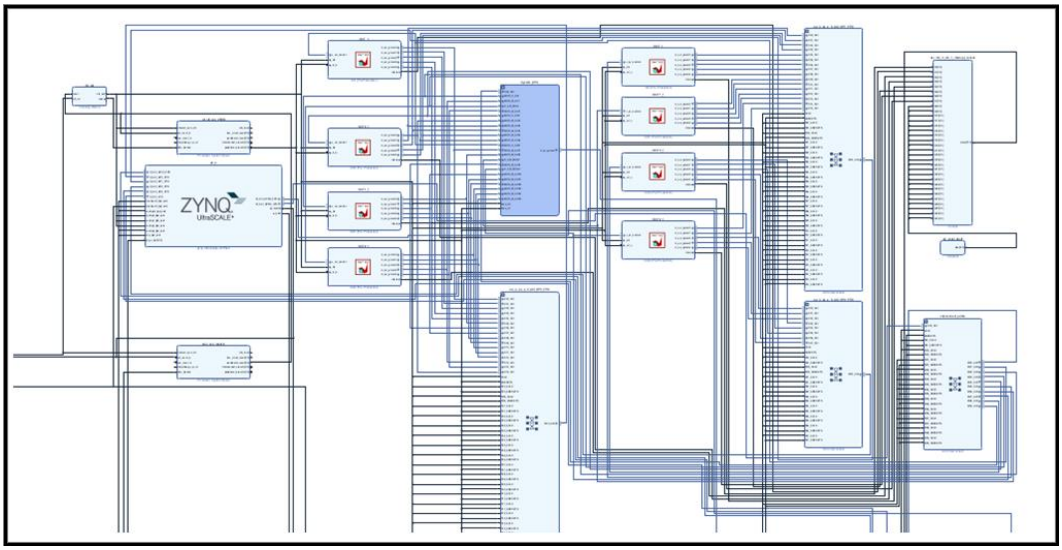

Figure 2. Synthesized system level design with accelerated transformation and quantization kernels.

The proposed methodology to perform encoder and decoder operations of transformation and quantization usually employed in video CODEC standards on a heterogeneous platform is compared in terms of resource utilization and throughput as shown below in table-1. The overall resource utilization of the proposed design is 60.36% out of the available total resource on the ZCU-104 board, which operates at 262.26MHz and total power consumption of 7.651watts.

Table 1: Comparison of Resource Utilization and Throughput

High level Design	Targeted FPGA	HLS optimization Technique used	LUTs	FFs	DSPs	RAM (Kbits)	Design operating Freq (MHz)	Throughput (in FPS)
2D-DQ & IT [18]	Xilinx Zynq ZC-702	Pipeline and Loop unroll	22.6K	18.8K	44	24	100	1080p @13
2D-DQ & IT [19]	XC7Z045	Pipeline, Loop unroll, Resource binding and Allocation	22.5k	18.8K	44	24	175	1080p @24

DOI: https://doi.org/10.54216/IJNS.270222

Forward & Inverse 2D-T & Q [17]	Intel Stratix 10 SX	Pipeline, Loop Unroll	593.9 K	1368.7 K	125 0	3565	243.75	1080p @47.6
Proposed Forward & Inverse 2D-T & Q	Xilinx Zynq ZCU-104	Dataflow, Pipeline, Loop Unroll	139.0 K	181.4K	232	241.5	262.26	1080p @30.3

The proposed design is benchmarked against prior works, focusing on the high-level design methodologies adopted for transform and quantization kernel development, as well as their corresponding FPGA-based evaluations. The designs presented in [18] and [19] implement inverse transform (IT) and dequantization (DQ) modules suitable for an HEVC video decoder. In [18], the functional software modules are synthesized into RTL using the Vivado HLS tool and integrated into the processing system via DMA. The resulting hardware-software co-designed system is deployed on the Xilinx Zynq ZC702 platform, which features a dual-core ARM Cortex-A9 processor and programmable logic. Experimental results report a throughput of 1080p at 13 frames per second (FPS). Similarly, the work in [19] presents both low-level RTL and high-level HLS implementations of the IT and DQ blocks. Their analysis demonstrates that the high-level implementation achieves lower resource utilization while maintaining a higher throughput of 1080p at 24 FPS. In contrast, the work in [17] adopts an OpenCL-based design methodology for end-to-end transformation, quantization, and their inverse operations like the scope of the proposed architecture. However, their implementation targets a high-end Intel Stratix-10 FPGA and includes additional modules for prediction, making it suitable for VVC codec applications. Despite the more advanced platform, the reported throughput for an 8×8 block size in [17] is 6.5 FPS, which is significantly lower than the 20 FPS achieved by the proposed design on the Xilinx ZCU104 board.

To verify throughput improvement by hardware acceleration on FPGA versus software implementation, the proposed design is implemented in a high-level programming language and executed solely on the ARM processing system of the ZCU-104 board. The software design is verified for functional correctness and throughput measurement. The execution time of this software implementation is then compared against the hardware-accelerated version running on the FPGA. Table 2 presents the throughput gains achieved by the hardware-accelerated kernel design on the FPGA, highlighting the performance improvement over the software-only execution on the ARM processor. The results demonstrate the effectiveness of the FPGA-based acceleration significantly improves processing throughput for computationally intensive video codec operations.

Table 2: Throughput analysis of Hardware vs Software implementations

Video class & Resolution	Video Sequence Name	Test Sequence Description/Feature	Hardware Execution time (in ms)	Software Execution time (in ms)	Throughput improvement in time by hardware Acceleration
UVG Dataset [24] 3840x2160	HoneyBee	A fixed camera focused on a honeybee flying between flowers. Fast-moving subject and Complex floral details in blurred foreground/Background	208	2776	13.34
B [23] 1920x1080	ParkScene	Natural Park footage, medium motion, good spatial variety.	50	596	11.92
	BQTerrace	Outdoor garden terrace scene with complex textures and lighting.			

DOI: https://doi.org/10.54216/IJNS.270222

C [23] 832x480	BasketballdrillText	Indoor sports training, quick player motion, camera zoom.	11	112	10.18
	PartyScene	Indoor party scene with variable lighting, skin tones, and movement.			
D [23] 416x240	BlowingBubbles	Children blowing bubbles, natural skin tones, subtle motion.	23	260	11.30
	BQSquare	Outdoor square scene, moving people and objects			
E [23] 1280x720	KristenandSara	Conversational scene, varying facial expressions	23	268	11.65
	FourPeople	Talking head scenario, static background, multi-person conferencing.			
F [23] 1024x768	China Speed	Screen content, synthetic graphics + camera input.	20	226	11.3

The quality of the decoded video frames across test sequences of varying resolutions is evaluated using objective metrics, specifically Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM), for Quantization Parameter (QP) values of 22, 27, 32, and 37. The quantization step sizes corresponding to these QP values are derived from a novel Neutrosophic-based quantization matrix, which is integrated into the proposed video codec architecture for both quantization and dequantization processes. Figures 3 and 4 illustrate the variation of PSNR and SSIM metrics for decoded frames at different QP settings and quantization type i.e NS based quantization and HVS based quantization. The perceptual quality of the decoded 2K HD video frames is high, with an average PSNR of 38 dB and an average SSIM of 0.9507. Furthermore, the PSNR and SSIM values obtained for test sequences across Classes C to F consistently exceed those achieved for higher resolution 2K and 4K sequences, indicating strong performance of the proposed CODEC across diverse content types and resolutions. The proposed design also achieves perceptually high-quality reconstruction, as evidenced by 3.77% PSNR improvement and 1.83% SSIM improvement over regular HVS based quantization.

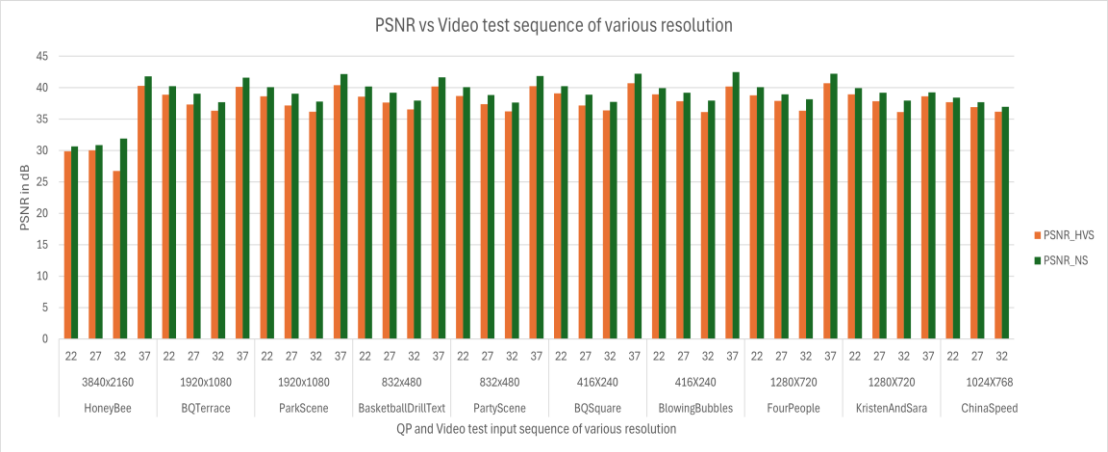

Figure 3. PSNR for NS and HVS based quantization vs test input sequence and QP values.

DOI: https://doi.org/10.54216/IJNS.270222

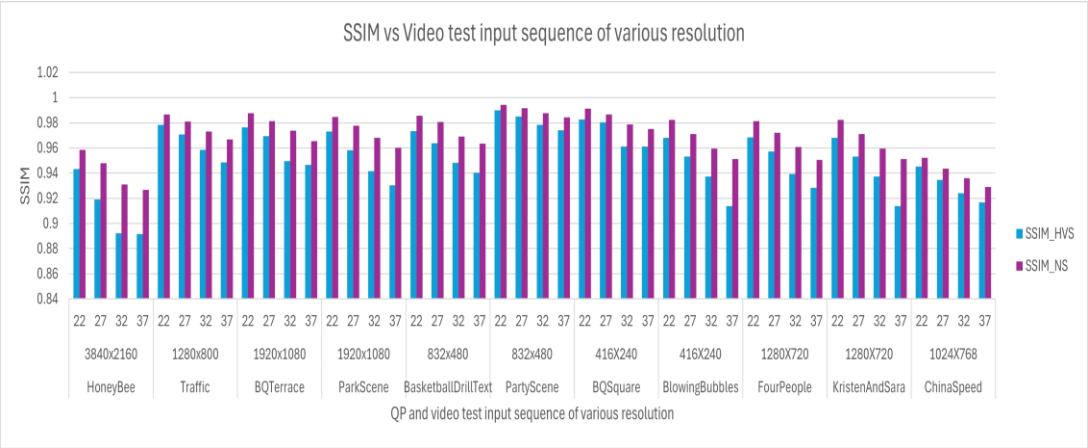

Figure 4. SSIM for NS and HVS based quantization vs test input sequence and QP values.

5. Conclusion

In the proposed work, a novel integer quantization matrix is derived using a Neutrosophic approach, incorporating key perceptual parameters such as display resolution and characteristics of the Human Visual System (HVS) to enhance visual quality. The design is implemented as a set of High-Level Synthesis (HLS) IPs using the OpenCL framework, along with the Vitis Vision Accelerated Library. This work presents a high-level design methodology for developing and evaluating hardware-accelerated transformation and quantization kernels on a heterogeneous computing platform featuring the Xilinx Zynq ZCU104 FPGA. Various HLS optimization techniques, including loop unrolling, pipelining, and dataflow optimization, are employed to maximize throughput on the FPGA as compared to the baseline software implementation running on the ARM processing system. Experimental results demonstrate a throughput improvement of approximately 13× for 4K resolution and 12× for 2K video sequences over the software-only implementation. The design also delivers perceptually high-quality reconstruction, as demonstrated by improvements of 3.77% in PSNR and 1.83% in SSIM metrics compared to HVS-based quantization. The complete end-to-end encoder and decoder system incorporates transformation, quantization, and their corresponding inverse operations, utilizing approximately 60% of the available FPGA resources and operating at a clock frequency of 262.26 MHz. The final implementation supports real-time performance, achieving a decoding rate of 1080p at 30 fps and 2160p at 8 fps on the target platform. Notably, the proposed architecture achieves higher throughput with smaller transform block sizes, while performance gradually decreases as block size increases. To maintain high throughput for larger transform blocks, these blocks should be internally partitioned into smaller units and processed in parallel through loop unrolling. This optimization will be explored in future work using the same test sequences and evaluation conditions.

Funding: "This research received no external funding"

Conflicts of Interest: "The authors declare no conflict of interest."

References

[1] K. Misra, A. Segall, and F. Bossen, "Tools for Video Coding Beyond HEVC: Flexible Partitioning, Motion Vector Coding, Luma Adaptive Quantization, and Improved Deblocking," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 5, pp. 1361–1373, May 2020, doi: 10.1109/TCSVT.2019.2945473.

[2] Y.-W. Huang *et al.*, "Block Partitioning Structure in the VVC Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 10, pp. 3818–3833, Oct. 2021, doi: 10.1109/TCSVT.2021.3088134.

[3] Ghanbari Talouki, A. Koochari, and S. A. Edalatpanah, "Applications of Neutrosophic Logic in Image Processing: A Survey," *J. Electr. Comput. Eng. Innov. (JECEI)*, vol. 10, no. 1, pp. 243–258, Jan. 2022, doi: 10.22061/jecei.2021.8069.474.

[4] S. Mandour, "An Exhaustive Review of Neutrosophic Logic in Addressing Image Processing Issues," *Neutrosophic Syst. Appl.*, vol. 12, pp. 36–55, Dec. 2023, doi: 10.61356/j.nswa.2023.110.

[5] A. Salama, M. A. I. Abdel-Khalek, F. Aripov, and S. Mohammed, "Neutrosophic Logic-Based Decision-Making Framework for Smart City Applications," *J. Ambient Intell. Humaniz. Comput*, vol. 12, no. 3, pp. 3421–3432, Mar. 2021, doi: 10.1007/s12652-020-02501-3.

DOI: https://doi.org/10.54216/IJNS.270222

[6] A. Salama, Z. Tarek, Y. Darwish, S. Elseuofi, and M. Y. Shams, "Neutrosophic Encoding and Decoding Algorithm for ASCII Code System," vol. 63, 2024, doi: 10.5281/zenodo.10531771.

[7] T. Alhussein, M. K. Hussain, and R. M. Ali, "A Review on Neutrosophic Sets and Their Applications in Image Processing," *Int. J. Comput. Appl.*, vol. 175, no. 1, pp. 18–25, Apr. 2021, doi: 10.5120/ijca2021921874.

[8] X. Zhao *et al.*, "Transform Coding in the VVC Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 10, pp. 3878–3890, Oct. 2021, doi: 10.1109/TCSVT.2021.3087706.

[9] M. Budagavi, A. Fuldseth, G. Bjøntegaard, V. Sze, and M. Sadafale, "Core Transform Design in the High Efficiency Video Coding (HEVC) Standard," *IEEE J. Sel. Topics Signal Process.*, vol. 7, no. 6, pp. 1029–1041, Dec. 2013, doi: 10.1109/JSTSP.2013.2270429.

[10] S. Shen, W. Shen, Y. Fan, and X. Zeng, "A Unified 4/8/16/32-Point Integer IDCT Architecture for Multiple Video Coding Standards," in *Proc. IEEE Int. Conf. Multimedia Expo*, Melbourne, VIC, Australia, 2012, pp. 788–793, doi: 10.1109/ICME.2012.7.

[11] J.-S. Park, W.-J. Nam, S.-M. Han, and S.-S. Lee, "2-D Large Inverse Transform (16×16, 32×32) for HEVC (High Efficiency Video Coding)," *JSTS J. Semicond. Technol. Sci.*, vol. 12, no. 2, pp. 203–211, Jun. 2012, doi: 10.5573/jsts.2012.12.2.203.

[12] P. K. Meher, S. Y. Park, B. K. Mohanty, K. S. Lim, and C. Yeo, "Efficient Integer DCT Architectures for HEVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 1, pp. 168–178, Jan. 2014, doi: 10.1109/TCSVT.2013.2276862.

[13] W. Hamidouche *et al.*, "Hardware-Friendly Multiple Transform Selection Module for the VVC Standard," *IEEE Trans. Consum. Electron*, vol. 68, no. 2, pp. 96–106, May 2022, doi: 10.1109/TCE.2022.3163345.

[14] L. Prangnell and V. Sanchez, "Adaptive Quantization Matrices for HD and UHD Resolutions in Scalable HEVC," in *Proc. Data Compression Conf. (DCC)*, Snowbird, UT, USA, 2016, p. 626, doi: 10.1109/DCC.2016.47.

[15] D. Grois and A. Giladi, "Perceptual quantization matrices for high dynamic range H.265/MPEG-HEVC video coding," *Proc. SPIE*, vol. 11137, p. 111370O, Feb. 2020, doi: 10.1117/12.2525406.

[16] N. Alquadami and S.-D. Kim, "OpenCL-based optimization methods for utilizing forward DCT and quantization of image compression on a heterogeneous platform," *J. Real-Time Image Process.*, vol. 12, no. 2, pp. 219–235, Aug. 2016, doi: 10.1007/s11554-015-0507-5.

[17] H. M. Waidyasooriya *et al.*, "OpenCL-Based Design of an FPGA Accelerator for H.266/VVC Transform and Quantization," in *Proc. IEEE 65th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, 2022, pp. 1–4.

[18] M. Kammoun, A. Ben Atitallah, K. M. A. Ali, and R. Ben Atitallah, "Case study of an HEVC decoder application using high-level synthesis: intraprediction, dequantization, and inverse transform blocks," *J. Electron. Imaging*, vol. 28, no. 3, p. 033010, 2019, doi: 10.1117/1.JEI.28.3.033010.

[19] Ben Atitallah, M. Kammoun, K. M. A. Ali, and R. Ben Atitallah, "An FPGA comparative study of high-level and low-level combined designs for HEVC intra, inverse quantization, and IDCT/IDST 2D modules," *Int. J. Circuit Theory Appl.*, pp. 1–17, 2020, doi: 10.1002/cta.2790.

[20] Pham-Quoc, "HLS and FPGA-Powered Streaming Video Encoder Accelerator for IoTs Edge Computing," *J. Adv. Inf. Technol.*, vol. 15, no. 1, pp. 59–65, Jan. 2024, doi: 10.12720/jait.15.1.59-65.

[21] *Vitis Vision Library User Guide*. [Online]. Available: https://docs.amd.com/r/en-US/Vitis_Libraries/vision/overview.html_1_3. [Accessed: 10 Jan. 2023].

[22] *ZCU104 Board User Guide*. [Online]. Available: https://docs.amd.com/v/u/en-US/ug1267-zcu104-eval-bd. [Accessed: 10 Jan. 2023].

[23] M. Xu, L. Jiang, X. Sun, Z. Ye, and Z. Wang, "Learning to Detect Video Saliency With HEVC Features," *IEEE Trans. Image Process.*, vol. 26, no. 1, pp. 369–385, Jan. 2017, doi: 10.1109/TIP.2016.2628583.

[24] Mercat, M. Viitanen, and J. Vanne, "UVG dataset: 50/120fps 4K sequences for video codec analysis and development," in *Proc. ACM Multimedia Syst. Conf.*, Istanbul, Turkey, Jun. 2020.

DOI: https://doi.org/10.54216/IJNS.270222