J Inno Sci Eng, 2025, 9(2):268-278

DOI:https://doi.org/ 10.38088/jise.1712080

Research Article

Speeding up the Discrete Cosine Transform Through Custom Operations and Fixed-Point Arithmetic

Latif Akçay [1], **Mustafa Alptekin Engin**[2*]

[1] *Erzurum Technical University, Department of Computer Engineering, 25050, Erzurum, Türkiye*

[2] *Bayburt University, Department of Electrical Electronics Engineering, 69000, Bayburt, Türkiye*

ARTICLE INFO

Received Date: 2/06/2025
Accepted Date: 31/07/2025

Cite this paper as:
Akçay, L., & Engin, M.A. (2025). Speeding up the Discrete Cosine Transform Through Custom Operations and Fixed-Point Arithmetic *Journal of Innovative Science and Engineering.* 9(2), 268-278.

*Corresponding author: Mustafa Alptekin Engin
E-mail:maengin@bayburt.edu.tr

Keywords:
Discrete cosine transform
Efficient processor design
Signal processing
Transport-Triggered architecture
Application-specific processor design

ABSTRACT

Digital signal processing applications are becoming increasingly important because modern systems work with much larger amounts of data than before. The Discrete Cosine Transform (DCT), used in almost all multimedia compression methods, creates a significant computational load especially in resource-constrained embedded systems. This study proposes four custom operations compatible with Transport-Triggered Architecture (TTA). To enhance computational efficiency and avoid floating-point overhead, fixed-point arithmetic is used. To analyse the effect of the proposed operations, different Application-Specific Instruction Set Processor (ASIP) configurations were created on a general-purpose processor architecture. Performance analyses show that speedups between 2x and 3.5x are achieved. In addition, the developed processor models have been implemented in hardware. FPGA synthesis results indicate a reasonable increase in chip area, showing that the proposed solutions could be an efficient alternative, particularly for limited-resource embedded systems.

1. Introduction

Today, applications based on digital signal processing (DSP) are used for much more than basic data processing tasks. They are employed in many different areas, including image and audio processing, biomedical signal analysis, communication systems, radar and sonar applications, machine fault detection, remote sensing, and artificial intelligence-supported visual recognition. The variety of applications has made DSP methods indispensable for data analysis and real-time decision support systems. Despite developments in software and hardware technologies, the need to efficiently represent, transmit, compress and interpret signals is increasing day by day. These applications, particularly those related to current topics such as artificial intelligence and machine learning, require faster and more energy-efficient hardware solutions as data volumes increase.

Many natural and engineered signals have complex structures that do not immediately reveal the important information they contain. Therefore, these signals need to be analysed in both the time and frequency domains. In this context, the discrete cosine transform (DCT), which was proposed by Ahmed, Natarajan and Rao (1974), stands out as an effective and widely used tool among signal processing methods operating in both domains [1]. The DCT enables data to be represented in a more meaningful and compressible form, particularly due to its ability to concentrate the energy of signals in fewer low-frequency components. Although its structure resembles that of the Fourier transform, the fact that it only works with real numbers makes the DCT more suitable for use with real-valued data. Despite decades having passed since its proposal, the DCT is still widely used today thanks to its advantages in energy concentration and information representation. In the field of image compression in particular, the DCT method forms the basis of the JPEG standard [2]. It is also one of the basic components of video compression standards such as MPEG, H.261 and H.263 [3]. Additionally, the DCT is employed in audio coding standards such as MP3 and AAC for audio compression [4]. It is also used in the feature extraction stage of face recognition systems, providing input for classification algorithms [5]. In biomedical signal processing, the DCT is particularly used in electrocardiogram (ECG) signal analysis to improve the signal-to-noise ratio and for feature extraction [6]. Furthermore, the DCT is favoured for adding hidden information to digital images in digital watermarking applications, which are used to protect copyright and ensure content integrity [7].

Over the years, hardware designs for the fast and efficient implementation of the DCT operations have attracted the attention of many researchers and enabled them to develop different architectural approaches. The tendency of researchers and the studies in the literature can be examined in several groups. Many studies have proposed hardware accelerator designs for the DCT and IDCT [8-9-10-11]. These designs have been implemented on FPGAs or ASICs, and efficient solutions that promise high performance and low power consumption have been developed. However, most of these studies are specific to a particular case and are not suitable for widespread use. This is because they are either a part of a digital signal processor (DSP) or are only loosely coupled to the data path of specific processors, and thus not flexible. Another approach is to extend the existing instruction sets with custom instructions [12-13-14-15-16]. Such designs must be compatible with the instruction set architecture (ISA) on which they are built. In addition, these solutions can only be implemented on certain processors, which creates a flexibility problem similar to pure hardware accelerators. Due to their parallel processing capabilities, Graphics Processing Units (GPUs) are well-suited to building high-speed DCT applications. As a matter of fact, a large number of investigations have been carried out with the objective of accelerating DCT on GPUs [17-18-19-20]. While these studies demonstrate high performance in their respective fields, the main issue is that they are not suitable for embedded systems.

When developing a processor for a specific embedded system application, designers should consider several criteria. Firstly, the proposed solution should be flexible and able to be implemented according to various priorities. However, the architecture should be suitable for providing low power consumption and for implementation in a narrow chip area. Due to possible budget constraints, the solution should also be free to use and distribute. The fact that the design environment and auxiliary tools are open source and reconfigurable will also provide serious advantages to designers, as they will be able to customise them to their needs. From this perspective, open-source application-specific instruction set processors (ASIPs) are the most suitable architectural option for embedded systems [21]. While there are commercial solutions for developing such processors [22-23], there are also free options available [24]. Examining the literature from this perspective reveals some notable [25-26-27-28]. Heikkinen et al. (2003) presented an ASIP for a 32-point DCT operation using the open-source MOVE framework [25]. While they did not present a custom operation design for the DCT operation itself, the authors were able to develop a high-performance processor running at 700 MHz by combining a large number of computational units with six parallel data paths. Shi et al. (2010) used multiplication and addition operations to accelerate 32-point DCT operations in their ASIP architecture, which was designed for MP3 audio processing [26]. Boutellier et al. (2011) incorporated custom ASIP processors for the DCT and the inverse DCT (IDCT) operations into their multiprocessor system-on-chip (MPSoC) design for MPEG video codecs [27]. However, this study did not provide any detailed performance results specifically for the DCT operations. In a later study in which some of the authors from the previous study also participated, Yviquel et al. (2013) introduced various network architectures for the MPSoC they designed, presenting a detailed performance analysis on different FPGAs [28].

The rest of the study is organised as follows: The second section presents the basic architecture, design environment, and processor development process used in the study step by step. The third section then evaluates the performance analysis and hardware implementation with simulations and tests. The whole study is concluded in the fourth section.

2. Material and Methods

In this section, we provide a brief explanation of the processor architecture and the development environment. In addition, we delve into our design approach.

2.1. Discrete Cosine Transform

The DCT is a transformation that allows a discrete signal to be expressed as the sum of cosine functions at different frequencies. It is widely used because it is an efficient and fast method in signal processing applications. The one-dimensional forward direction (1D-FDCT) equation of the DCT is given in Equation 1.

$$X[k] = \alpha(k) \sum_{n=0}^{N-1} x[n] cos\left(\frac{\pi(2n+1)k}{2N}\right) \quad (1)$$

In Equation 1, x[n] is the input signal of length N, and X[k] represents the frequency component after the transformation where (k=0,1,…,N-1). Here, the normalization coefficient α(k) is defined in Equation 2.

$$\alpha(k) = \begin{cases} \sqrt{\frac{1}{N}}, & k = 0 \\ \sqrt{\frac{2}{N}}, & 1 \leq k \leq N-1 \end{cases} \quad (2)$$

From the frequency components obtained as a result of the FDCT transformation, we can obtain the original signal via the inverse transformation (1D-IDCT) with the equation given in Equation 3.

$$x[n] = \sum_{k=0}^{N-1} \alpha(k) X[K] cos\left(\frac{\pi(2n+1)k}{2N}\right) \quad (3)$$

The 2D forward DCT operation (2D-FDCT), which is used especially in image processing applications, is performed by applying the 1D-FDCT operation in both row and column directions, as specified in Equation 4.

$$X[u][v] = \alpha(u)\alpha(v) \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} A[m][n] cos\left(\frac{\pi(2m+1)u}{2N}\right) cos\left(\frac{\pi(2n+1)v}{2N}\right) \quad (4)$$

In Equation 4, A[m][n] represents the input matrix of N×N size, and u and v represent the frequency indices in the row and column directions, respectively. Similarly, we can obtain the original signal via the inverse transform (2D-IDCT) with the equation specified in Equation 5.

$$A[m][n] = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} \alpha(u)\alpha(v) X[u][v] cos\left(\frac{\pi(2m+1)u}{2N}\right) cos\left(\frac{\pi(2n+1)v}{2N}\right) \quad (5)$$

2.2. Transport-Triggered Architecture

Unlike traditional processor designs, Transport-Triggered Architecture (TTA) is an innovative processor design method that ties the computational approach to the movement of data between blocks [24]. In the TTA approach, instructions are solely responsible for transferring data between functional units (FUs). In classical processor architectures, such as Reduced Instruction Set Computing (RISC), however, instructions are coded according to the operation to be performed. The only defined command in TTA is the 'move' command, which specifies the destination of the data. When relevant data arrives at a FU and is written to the trigger port, the calculation begins, and the result is written to the output ports. The result of this process is then transferred to the next FU. As can easily be predicted, the most important result of this approach is a reduction in control complexity and more energy-efficient processing.

The origins of TTA can be traced back to dataflow computing and Very Long Instruction Word (VLIW) architectures developed in the 1980s [29]. In 1997, Henk Corporaal systematised the concept of TTA and introduced it to academic literature [24]. Over the subsequent years, the architecture's practical applications have been tested in various fields using both FPGA- and ASIC-based prototypes [30-31]. It has continued to attract attention in recent years as an alternative ASIP architecture in popular areas such as digital signal processing (DSP), soft-core embedded processor design, and cryptography [32-33-34-35-36].

One of the major benefits offered by TTA is instruction-level parallelism (ILP). Multiple operations can be triggered simultaneously on different FUs with independent data paths. Furthermore, the customisation of FUs and connection networks (transport bus interface) provides designers with significant flexibility for tasks focused on chip area, performance, or power consumption. Therefore, TTA offers an excellent foundation for ASIP designs. Figure 1 shows the structural form of a typical TTA processor.

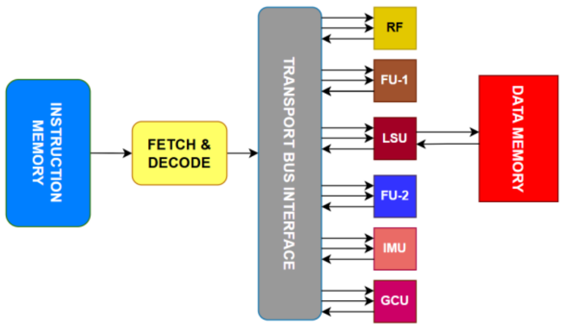

Figure 1: Principle block design of TTA processors.

2.3. Design and Development Environment

OpenASIP, formerly known as TTA-Based Co-Design Environment (TCE), is an integrated toolset design and implementation of TTA processors [37]. The reason for this name change is that the toolset has been given a more general name, along with support for other instruction sets such as RISC-V in addition to TTA [38]. OpenASIP has been developed at the Tampere University as an open source and freeware. Toolset offers a complete design and development environment, starting from an application written in high-level programming languages to the production of synthesizable processor hardware. Designers can customize the FUs, register files (RFs), or interconnection networks of template processors according to project requirements, or create their own models. In addition to the standard operations offered by OpenASIP, application-specific operations can also be added to the compilation chain, and new hardware descriptions can be added to the hardware database (HDB) for customised FUs. Cycle-accurate simulations can be run on the prepared model, profiling analyses can be performed, and finally hardware description language (HDL) code can be generated for the entire processor design. Figure 2 shows the design phases of a TTA processor in the OpenASIP environment. Template architectures are prepared using a visualized Processor Designer (ProDe) tool. TCE C Compiler (TCECC) compiles high-level applications along with an architecture definition file (ADF). The Proxim tool is responsible for all simulation and profiling analyses. The Operation Set Editor (OSEd) tool defines and manages machine-level operations. Finally, the hardware implementation of the target processor model is generated using the Processor Generator (ProGe) tool.

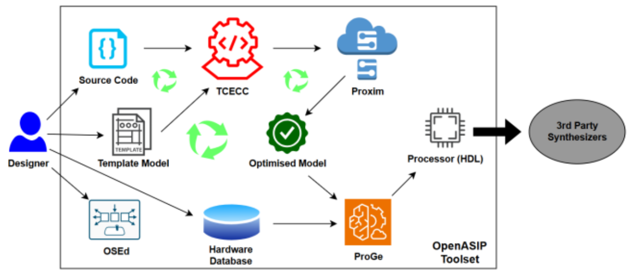

Figure 2: Typical design process of a TTA core in OpenASIP toolset.

2.4. 2D DCT / IDCT Application and Template Architecture

As stated in Section 2.1, the 2D-DCT/IDCT method comprise a high number of intensive multiplication and addition operations. However, it should be noted that the calculations also contain many floating-point arithmetic operations. However, native support for these operations requires a complex and chip-area-demanding floating-point unit (FPU), which is unsuitable for low-power embedded systems. On the other hand, compiler-level emulation of floating-point arithmetic requires performing numerous inefficient integer operations. In this case, fixed-point arithmetic can be used as an alternative. However, this method introduces computational errors due to potential overflows and limited dynamic range. Therefore, it is crucial to choose a fixed-point arithmetic format that provides negligible precision errors for the DCT implementation. In addition, the chosen format directly determines memory usage and also the number of machine-level operations to be performed. Following our analysis and simulations under these conditions, we opted for the Q15.16 (Kim et al., 1998) format in this study, balancing accuracy and computational efficiency [39]. Accordingly, a C application that operates 2D-DCT/IDCT algorithms on grey-scale images and uses fixed-point arithmetic without requiring floating-point support has been developed. The DCT coefficients were converted to Q15.16 format and recorded as a constant matrix. Since these values are fixed coefficients, they were later stored in local memories within the design. The developed software application accepts images in the 128x128 integer format and subsequently converts the pixel values back to integer format following the application of the DCT and IDCT operations.

In the processor design process, a template architecture was selected for the initial profiling analysis of the 2D-DCT/IDCT application. In accordance with the objective of the study, a general-purpose template model that supports minimal and 64-bit operations presented in the OpenASIP tool has been selected. Figure 3 illustrates the Processor Designer (ProDe) model of the template processor. The load-store unit (LSU) of the model incorporates basic memory operations. The arithmetic logic unit (ALU) is capable of processing 23 simple instructions, including ADD, OR, AND, and SHIFT. In addition to these components, a separate FU incorporates 64-bits integer multiplication (MUL) and multiply–accumulate (MAC) operations. The RF block is a single-port register file that contains 64-bits wide 16 registers, while control operations are implemented in the GCU block.

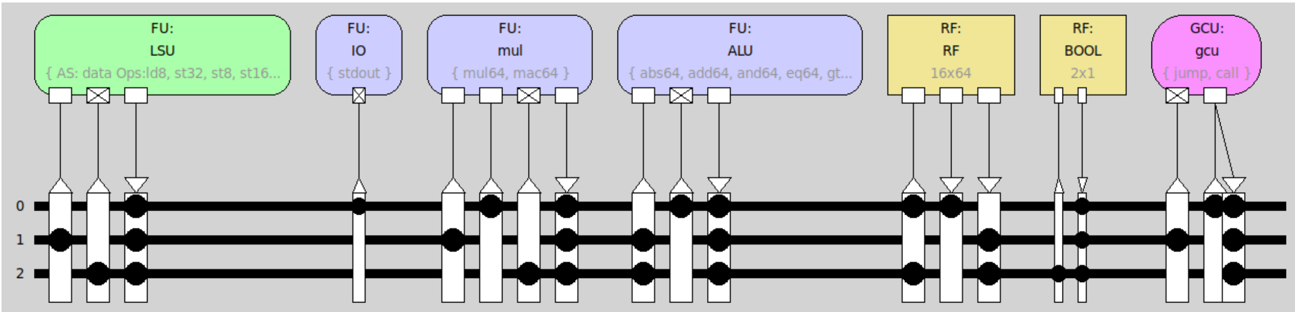

Figure 3: Typical design process of a TTA core in OpenASIP toolset.

2.5. Custom DCT Operations

When 2D-DCT/IDCT algorithms are examined in depth, it can be easily seen that pixel values associated with constant coefficients are continuously multiplied and the result is formed by adding these multiplications. However, when this process is applied with general purpose instructions for each 8x8 pixel block, it requires intensive memory and arithmetic operations, which prolongs the execution time of the algorithm. Therefore, custom DCT operations that effectively reduce computation time have been developed in this study.

As the DCT coefficient matrix can be stored in a fixed and small-sized register array, it is not necessary to read it from the memory each time. Instead, storing the coefficients in the Q15.16 format in a small local memory within a FU reduces the need for memory access rate and provides performance gains. Thus, we implemented a 64x32-bit block memory (BRAM) to store precomputed DCT coefficients. The width of this memory can be changed according to the DCT format used in other studies. The custom operation we call multiply-and-accumulate operation (MACOP) is triggered when pixel values are read from memory. An additional port transmits the information that the vertical or horizontal values of the coefficient matrix will be read to the address decoder of the local memory. In the subsequent reading phase, the values in the following column or row of the matrix are assigned to the memory output. Therefore, the coefficients are prepared in advance and multiplied in parallel with the pixel values transferred from the main memory. The products are then summed to produce the result. Figure 4 demonstrates the computation architecture of the MACOP operation.

Another custom operation we designed for 2D-DCT/IDCT application is CLAMP. The purpose of this operation, which is not actually a part of the algorithm, is to keep the values within the representable range. This operation simply ensures that the pixel values exceeding the limit are assigned as 0 or 255 during the image reconstruction operation.

One of the computational bottlenecks of our 2D-DCT/IDCT application, developed using Q15.16 fixed-point arithmetic, is the conversion of integers to fixed-point numbers and vice versa. The SHIFT operation in the minimal template is a 1-bit shift to the right or left. The Q15.16 format requires a 16-bit shift operation for each integer to fixed-point format or fixed-point to integer conversion, which means waiting at least 16 clock cycles for each coefficient used in the DCT/IDCT calculation. Therefore, we designed custom operations for these conversions and named them INT2FIX and FIX2INT. These operations are designed to perform the conversion between both representations in a single clock cycle.

In the OpenASIP toolkit, predefined operations (AND, ADD, SHIFT, etc.) and custom operations developed by users are managed by the OSEd tool. We introduced the custom operations to OSEd and specified the input-output port features. We then defined and compiled the functions of the operations. We ran the necessary validation tests using the tool provided by OSEd to verify that the operations performed as the designer had expected. Consequently, the custom operations were integrated into the compilation chain and prepared for use in the 2D-DCT/IDCT application.

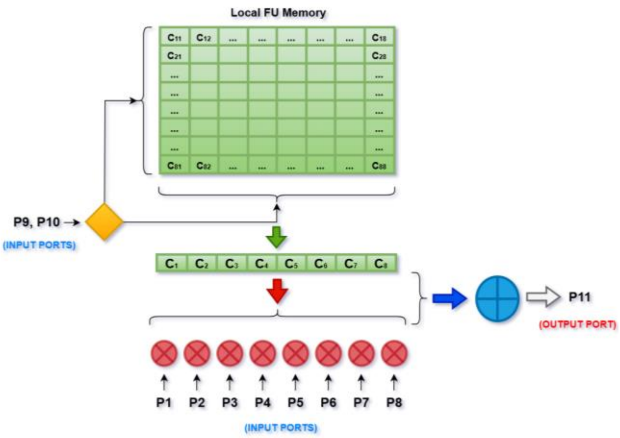

Figure 4: Design of the MACOP custom operation.

2.6. ASIP Design

The ProDe tool enables users to customise template processor models or create their own. They can add new functional units (FUs), change the structure of existing ones, or delete them entirely. They can also easily adjust the number of transport buses, the transport bus connection architecture, and the RF design. We developed our ASIP designs by integrating custom operations onto a minimal template configuration. This aimed to simplify ASIP architectures while also allowing us to analyse the performance contribution of custom operations and their impact on hardware resource utilisation without considering additional design factors. Thus, we first created our core ASIP architecture, DCT-P1, using the following steps: First, we removed the FU, which includes the MUL and MAC operations in the minimal template, from the design because we had already developed the MACOP operation. As this operation has 10 input ports and one output port, it could not be integrated with the other FUs in the architecture. Therefore, a new FU named 'DCT' was created, to which the MACOP operation was assigned. As the port numbers of the other custom operations (CLAMP, INT2FIX, FIX2INT) are suitable for the existing FUs, these were integrated into the ALU FU. After completing the integration of custom operations, the DCT-P1 FU was integrated into the system with its input ports connected to a single transport bus, similar to the other FUs in the minimal template, as seen in the DCT-P1 configuration shown in Figure 5. Similarly, the RF block was left with 16 64-bit registers.

In order to evaluate the performance gains of custom operations when using different transport bus architectures, we developed three configurations: DCT-P2, DCT-P3, and DCT-P4. This allowed us to observe how performance gains provided by custom operations would vary across different architectures, and to identify the optimal one. Figure 6, Figure 7, and Figure 8 demonstrate DCT-P2, DCT-P3, and DCT-P4 configuratios, respectively. In the DCT-P2 model, each input port of the DCT FU is connected to two transport buses, while in the DCT-P3, this unit is fully connected to the entire network. On the other hand, in the DCT-P4 configuration, the input and output ports of all FUs are fully-connected to the network.

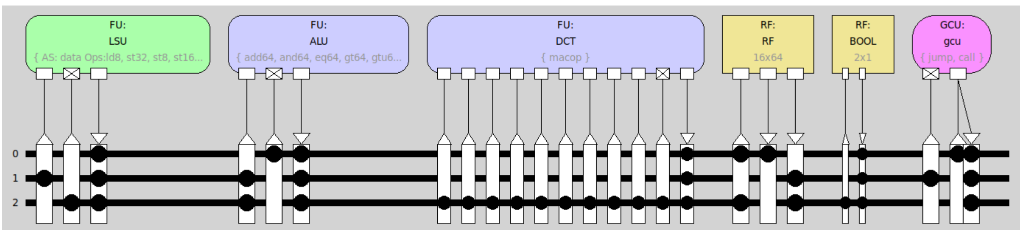

Figure 5: Structral model of the DCT-P1 processor including the DCT FU where custom operations reside.

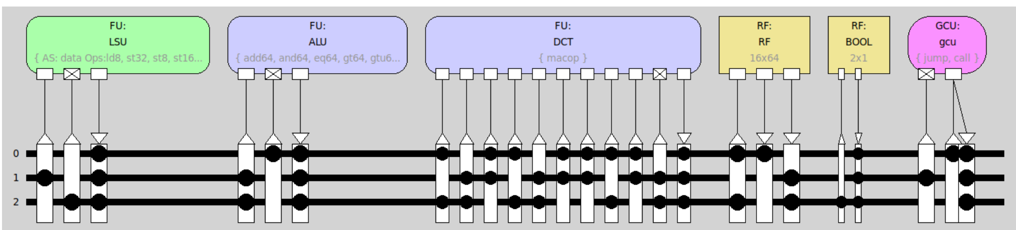

Figure 6: The ProDe model of the custom processor DCT-P2 derived from DCT-P1

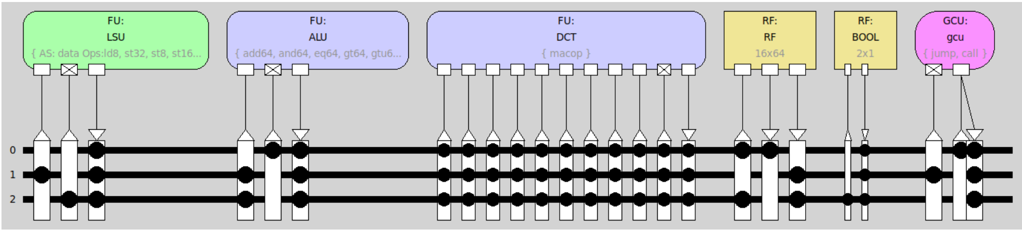

Figure 7: The ProDe model of the custom processor DCT-P3 with a fully-connected DCT-FU.

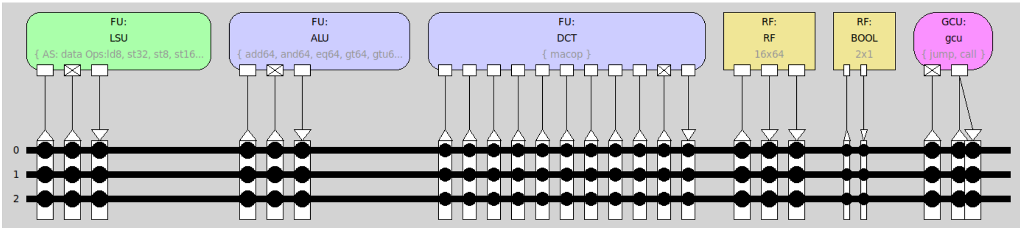

Figure 8: The ProDe model of the custom processor DCT-P4 with fully-connected bus architecture

3. Results and Discussion

In this section, we examine our ASIPs in different configurations in terms of performance and hardware resources.

3.1. Performance Analysis

Proxim is a practical tool that can be used to analyse cycle-accurate simulations and resource usage statistics in the OpenASIP environment. Proxim supports not only the standard operations offered by OpenASIP but also custom operations integrated by users. In order to utilise these useful capabilities, the 2D-DCT/IDCT application was run separately on the DCT-P1, DCT-P2, DCT-P3, and DCT-P4 cores. The performance data obtained for a 128x128 pixel grey-scale image is given in Table 1 in terms of total number of clock cycles (NoC). Instruction word length (IWL) for each configuration is given in the third coloumn.

In addition to the consideration of transport bus architectures and custom operations, another key issue to be addressed in performance analysis is the number of registers. A larger RF block theoretically provides the processor with more space for data storage, allowing it to execute fewer no-operation (NOP) instructions and spend less time waiting. Consequently, the running application is completed in less time. For all compared configurations in Table 1, a 16x64 RF, which is equivalent to the minimal template processor, was utilised. However, in order to evaluate the effect of the RF size, each configuration was also analysed with a 32x64 RF. This approach enables a comparative analysis of the impact of RF size across our custom cores.

Table 1 provides an analysis of the impact of RF size and architectural optimisations, such as custom operations and connection types of the transport buses in terms of NoC. Clearly, all DCT-P series processors significantly outperform the underlying 16x64 RF-sized minimal template processor, running the 2D-DCT/IDCT application at 3,480,576 cycles. Even the simplest DCT-P1 configuration (16x64 RF) doubles performance, running at a speed 2.04 times faster than the template, with 1,706,240 cycles. Increasing the RF size from 16x64 to 32x64 in the same DCT-P1 architecture results in an additional improvement of 6.6%, with 1,593,856 cycles.

The more advanced DCT-P2 and DCT-P3 configurations further highlight the impact of the optimisations. Specifically, the DCT-P2 (32x64) configuration offers a 66% improvement on the template, with 1,186,048 cycles and a 2.93x speedup. The increase in RF size also makes a noticeable contribution. For the DCT-P3 (32x64) configuration, the cycle count decreases to 1,182,208 and the improvement factor is 2.94. Although the DCT-P2 and DCT-P3 configurations offer similar performance, the DCT-P3 configuration is slightly more efficient on a per-cycle basis.

The highest performance is achieved with the DCT-P4 (32x64) configuration. This structure complets the application in just 982,528 cycles, making it 3.54 times faster than the template. This is the lowest NoC value among all configurations. It is also noteworthy that increasing the RF size from 16x64 to 32x64 yields a 24.4% performance increase (from 1,299,968 to 982,528 cycles) on this architecture. However, the RF size increase does not appear to contribute to the same extent across all configurations (11% for DCT-P2, 9% for DCT-P3). This demonstrates that the extent to which architectures utilise RF efficiently is also a significant variable.

Table 1: Information about Karasu River Kırkgöze gauging.

Processor	RF Size	IWL	NoC	Gain
Template	16x64	64	3,480,576	1
Template	32x64	67	3,019,776	1.15
DCT-P1	16x64	65	1,706,240	2.04
DCT-P1	32x64	67	1,593,856	2.18
DCT-P2	16x64	66	1,334,784	2.61
DCT-P2	32x64	68	1,186,048	2.93
DCT-P3	16x64	66	1,303,808	2.67
DCT-P3	32x64	68	1,182,208	2.94
DCT-P4	16x64	72	1,299,968	2.68
DCT-P4	32x64	73	982,528	3.54

3.2. Hardware Implementation

The OpenASIP toolkit allows users to generate synthesizable HDL code for their own processor designs. This is achieved using two separate tools. The Hardware Database Editor (HDBEditor) enables users to add new hardware designs and modify predefined HDL codes. As its name suggests, the ProGe tool generates hardware implementations of the ProDe model of a TTA core. The generated HDL codes are then converted into real chips using FPGA or ASIC synthesis tools. Following the same method, hardware implementations for custom operations and FUs were first prepared and then defined in HDBEditor in this study. Finally, ProGe was used to produce HDL codes for our ASIPs. We used Xilinx Vivado IDE [40], for hardware synthesis and preferred an FPGA part number xc7ac100t [41] as a hardware platform. Table 2 provides an analysis of the FPGA resources required for the minimal template core, DCT-P1 and DCT-P4 ASIPs. No separate hardware implementation was made for DCT-P2 and DCT-P3; therefore, they are not presented in Table 2. However, it can easily be said that due to the complexity of the transport bus connections of these processors, which have exactly the same FUs as DCT-P1 and DCT-P4, their hardware resource utilisation will be between these two processors. Since DCT coefficients are stored in the DCT-FU within the MACOP operation, it is seen from the Table 2 that there is a 64x32-bit BRAM resource in ASIP processors that is not in the template configuration. As expected, LUT and FF resource utilisation is higher in ASIPs. The DCT-P1 occupies 1.19 times more chip area than the template configuration in terms of LUT count, while this figure is 1.24 times more for FFs. However, due to the DCT-P4's more complex transport bus architecture, it requires slightly more chip area than the DCT-P1 in both respects. While there is no significant difference between the DCT-P1 and the template configuration in terms of LUT count, the DCT-P4 has 40% more registers than the template configuration, particularly in terms of FFs.

Our ASIP designs can operate at higher clock frequencies than the minimal template configuration. As can be easily seen from Table 2, a 13.6% higher maximum clock frequency is achieved on the same FPGA. This is because the DCT-P series lacks the 64-bit multiplication and multiply-add operations (mul and mac), which are present in the minimal template. As emphasized in Section 2.6, the custom MACOP operation, which extends the critical path, eliminates the need for mul and mac operations, so these operations were removed from the ASIPs. Thanks to the MACOP operation, the multiplication and addition operations in DCT calculations can be performed more efficiently.

Table 2: FPGA resource utilisation for the template and custom cores.

Processor	BRAM	LUT	FF	f_{max} (MHz)
Template	-	3,472	2,238	110
DCT-P1	1	4,135	2,785	125
DCT-P4	1	4,258	3,145	125

4. Conclusion

Digital signal processing (DSP) applications are one of the most critical areas forming the basis of modern technology and enabling more complex systems. Similarly, the importance of the discrete cosine transform (DCT), one of the most widely used DSP operations, has remained high for many years. The increasing need to compress high-resolution images necessitates the efficient use of this technique. ASIPs play a key role in the rapid and efficient execution of specific applications within embedded systems where space and energy resources are limited. TTA is a prominent method in ASIP architectures thanks to its open-source nature, free usage rights, highly functional design environment, and advanced customisation possibilities. In this study, we propose TTA-compatible custom operations that significantly accelerate 2D-DCT/IDCT computations. Since a high-precision FPU is expensive hardware for embedded processors, we preferred using fixed-point arithmetic while developing the C application, which basically takes grey-scale images as input and applies 2D DCT and IDCT operations, respectively. To provide high-level accuracy while keeping computational efficiency, we used Q15.16 fixed-point arithmetic. Profiling analyses were performed using the tools provided by the OpenASIP toolkit to develop a custom operation that optimises the multiplication and addition operations that create bottlenecks in DCT computations. Special shift operations were also developed to enable conversion between integer and fixed-point arithmetic, with the aim of improving overall performance. We developed a base ASIP processor, namely the DCT-P1, using the custom operations. We then prepared various configurations using different transport bus connection architectures and larger RF blocks. This enabled us to analyse the effect of the proposed custom operations under different scenarios. Our detailed performance analysis clearly demonstrates that our ASIP design achieves a performance gain between 2x and 3.54x. We also implemented our processor designs in hardware and shared the hardware resource analysis after synthesising on an FPGA. Comparative resource utilsations show that our ASIP design sligthly increases the maximum clock frequency while increasing the required slice registers and LUTs at rates ranging from 19% to 40%. Although there are numerous studies in the literature on DCT optimisations, this work proposes open-source and free TTA architecture-compatible, hardware-friendly operations that provide significant performance improvements. The presented ASIP designs can also be used as preprocessors. As a continuation of this study, we plan to develop a more comprehensive JPEG preprocessor using the custom operations developed in this study.

Article Information

Financial Disclosure This study was supported by Bayburt University Scientific Research Projects Coordination Unit under project number 2023/69002-01.

Authors' Contrtibution: All authors contributed equally to this work.

Conflict of Interest/Common Interest: No conflict of interest or common interest has been declared by the authors.

Ethics Committee Approval: This study does not require ethics committee permission or any special permission.

References

[1] Ahmed, N., Natarajan, T., & Rao, K. R. (1974). Discrete cosine transform. *IEEE Transactions on Computers. Institute of Electrical and Electronics Engineers*, *C–23*(1), 90–93. https://doi.org/10.1109/t-c.1974.223784

[2] Pennebaker, W. B., & Mitchell, J. L. (1992). *JPEG: Still Image Data Compression Standard*. Springer.

[3] Furht, B. (2008). MPEG-2 Video Compression. In *Encyclopedia of Multimedia* (pp. 446–448). Springer US.

[4] Brandenburg, K. (1999). MP3 and AAC explained. Proceedings of the AES 17th International Conference on High Quality Audio Coding.

[5] Kusnadi, A., Pane, I. Z., & Tobing, F. A. T. (2025). Enhancing facial recognition accuracy through feature extractions and artificial neural networks. *IAES International Journal of Artificial Intelligence (IJ-AI)*, *14*(2), 1056. https://doi.org/10.11591/ijai.v14.i2.pp1056-1066

[6] Singh, A. K., & Krishnan, S. (2023). ECG signal feature extraction trends in methods and applications. *Biomedical Engineering Online*, *22*(1), 22. https://doi.org/10.1186/s12938-023-01075-1

[7] Varghese, J., Bin Hussain, O., Subash, S., & T, A. R. (2023). An effective digital image watermarking scheme incorporating DCT, DFT and SVD transformations. *PeerJ. Computer Science*, *9*, e1427. https://doi.org/10.7717/peerj-cs.1427.

[8] Duspara, A., Kovac, M., & Mlinaric, H. (2021). Discrete cosine transform hardware accelerator in parallel ultra-low power system. *2021 International Symposium ELMAR. IEEE.*

[9] Pastuszak, G. (2015). Hardware architectures for the H.265/HEVC discrete cosine transform. *IET Image Processing*, *9*(6), 468–477.

[10] Chen, J., Liu, S., Deng, G., & Rahardja, S. (2019). Hardware efficient integer discrete cosine transform for efficient image/video compression. *IEEE Access: Practical Innovations, Open Solutions*, *7*, 152635–152645.

[11] Shabiul Islam, M., Salim Beg, M., Bhuyan, M. S., & Othman, M. (2006). Design and implementation of discrete cosine transform chip for digital consumer products. *IEEE Transactions on Consumer Electronics*, *52*(3), 998–1003. https://doi.org/10.1109/tce.2006.1706499

[12] Nguyen, H., & John, L. K. (1999). Exploiting SIMD parallelism in DSP and multimedia algorithms using the AltiVec technology. *Proceedings of the 13th International Conference on Supercomputing*.

[13] Lee, R. B., Fiskiran, A. M., Shi, Z., & Yang, X. (2003). Refining instruction set architecture for high-performance multimedia processing in constrained environments. *Proceedings IEEE International Conference on Application-Specific Systems, Architectures, and Processors*.

[14] Hwang, Y.-T., & Huang, T.-H. (2012). Efficient TWIN-VQ audio decoder implementation on a configurable processor using instruction extension. *2012 IEEE International Symposium on Circuits and Systems*.

[15] Kamal, M., Ghasemazar, A., Afzali-Kusha, A., & Pedram, M. (2014). Improving efficiency of extensible processors by using approximate custom instructions. *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014*.

[16] Zhang, Y., Ou, H., & He, J. (2022). JPEG compression coding optimization based on NEON instruction. In F. Zhao (Ed.), *5th International Conference on Computer Information Science and Application Technology (CISAT 2022)*. SPIE.

[17] Haweel, R. T., El-Kilani, W. S., & Ramadan, H. H. (2016). Fast approximate DCT with GPU implementation for image compression. *Journal of Visual Communication and Image Representation*, *40*, 357–365. https://doi.org/10.1016/j.jvcir.2016.07.003

[18] Chiper, D. F., & Dobrea, D. M. (2024). A novel low-complexity and parallel algorithm for DCT IV transform and its GPU implementation. *Applied Sciences (Basel, Switzerland)*, *14*(17), 7491. https://doi.org/10.3390/app14177491

[19] Agha, S., Jan, F., Khan, H. A., Kaleem, M., & Khan, M. (2024). Efficient motion estimation and discrete cosine transform implementation using the graphics processing units. *PloS One*, *19*(8), e0307217. https://doi.org/10.1371/journal.pone.0307217

[20] Yang S., Wang Y., Li L, Qin J, Bi G. (2025). Implementation and Optimization of 8×8 Block Discrete Cosine Transform on MGPUSim. *IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, 832–839.

[21] Keutzer, K., Malik, S., & Newton, A. R. (2003). From ASIC to ASIP: the next design discontinuity. *Proceedings. IEEE International Conference on Computer Design: VLSI in Computers and Processors*.

[22] Ezer, G. A. (2005). Breaking the I/O bottleneck for high-compute performance processing with Xtensa LX configurable and extensible processor architecture. In S. Sudharsanan, V. M. Bove Jr, & S. Panchanathan (Eds.), *Embedded Processors for Multimedia and Communications II*. SPIE.

[23] Shahabuddin, S., Mammela, A., Juntti, M., & Silven, O. (2021). ASIP for 5G and Beyond: Opportunities and Vision. *IEEE Transactions on Circuits and Systems. II, Express Briefs: A Publication of the IEEE Circuits and Systems Society*, *68*(3), 851–857. https://doi.org/10.1109/tcsii.2021.3050785

[24] Corporaal, H. (1997). Microprocessor architectures: From VLIW to TTA. John Wiley & Sons.

[25] Heikkinen, J., Sertamo, J., Rautiainen, T., & Takala, J. (2003). Design of transport triggered architecture processor for discrete cosine transform. *15th Annual IEEE International ASIC/SOC Conference*.

[26] Shi, Z., Qiu, P., Wang, Y., Wang, S., & Guo, W. (2010). A Sub-band Synthesis Filter parallel processor based on Transport Trigger

Architecture. *2010 3rd International Congress on Image and Signal Processing*.

[27] Boutellier, J., Silven, O., & Raulet, M. (2011). Automatic synthesis of TTA processor networks from RVC-CAL dataflow programs. *2011 IEEE Workshop on Signal Processing Systems (SiPS)*.

[28] Yviquel, H., Boutellier, J., Raulet, M., & Casseau, E. (2013). Automated design of networks of transport-triggered architecture processors using dynamic dataflow programs. Signal Processing. *Image Communication*, *28*(10), 1295–1302. https://doi.org/10.1016/j.image.2013.08.013

[29] Fisher, J. A. (1983). Very Long Instruction Word architectures and the ELI-512. Proceedings of the 10th Annual International Symposium on Computer Architecture - ISCA '83.

[30] Hamalainen, P., Heikkinen, J., Hannikainen, M., & Hamalainen, T. D. (2005). Design of transport triggered architecture processors for wireless encryption. *8th Euromicro Conference on Digital System Design (DSD'05)*.

[31] Guo, J., Dai, K., & Wang, Z. (2006). A heterogeneous multi-core processor architecture for high performance computing. In *Advances in Computer Systems Architecture* (pp. 359–365). Springer Berlin Heidelberg.

[32] Akcay, L., & Ors, B. (2021). Custom TTA operations for accelerating kyber algorithm. *2021 13th International Conference on Electrical and Electronics Engineering (ELECO)*.

[33] Akçay, L., & Yalçın, B. Ö. (2022). Analysing the potential of transport triggered architecture for lattice-based cryptography algorithms. *International Journal of Embedded Systems*, *15*(5), 404. https://doi.org/10.1504/ijes.2022.127164

[34] Akçay, L., & Yalçın, B. Ö. (2024). Lightweight ASIP design for lattice-based post-quantum cryptography algorithms. *Arabian Journal for Science and Engineering*. https://doi.org/10.1007/s13369-024-08976-w

[35] Safarpour, M., Hautala, I., Bordallo López, M., & Silvén, O. (2019). Transport triggered array processor for vision applications. In *Lecture Notes in Computer Science* (pp. 361–372). Springer International Publishing.

[36] Jääskeläinen, P., Tervo, A., Vayá, G. P., Viitanen, T., Behmann, N., Takala, J., & Blume, H. (2018). Transport-triggered soft cores. *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 83–90.

[37] Jääskeläinen, Pekka, Viitanen, T., Takala, J., & Berg, H. (2017). HW/SW co-design toolset for customization of exposed datapath processors. In *Computing Platforms for Software-Defined Radio* (pp. 147–164). Springer International Publishing.

[38] Armstrong, A., Bauereiss, T., Campbell, B., Reid, A., Gray, K. E., Norton, R. M., Mundkur, P., Wassell, M., French, J., Pulte, C., Flur, S., Stark, I., Krishnaswami, N., & Sewell, P. (2019). ISA semantics for ARMv8-a, RISC-v, and CHERI-MIPS. *Proceedings of the ACM on Programming Languages*, *3*(POPL), 1–31. https://doi.org/10.1145/3290384

[39] Kim, S., Kum, K.-I., & Sung, W. (1998). Fixed-point optimization utility for C and C++ based digital signal processing programs. *IEEE Transactions on Circuits and Systems II Analog and Digital Signal Processing*, *45*(11), 1455–1464. https://doi.org/10.1109/82.735357

[40] Chakraborty, S. (2017). Vivado Design Tools. In *Designing with Xilinx® FPGAs* (pp. 17–21). Springer International Publishing. https://doi.org/10.1109/access.2019.2947269

[41] Modi, H., & Athanas, P. (2015). In-system testing of Xilinx 7-Series FPGAs: Part 1-logic. *IEEE Military Communications Conference MILCOM 2015*.