Designing and Implementing a 2D Integer DCT Hardware Accelerator Fully Compatible with Versatile Video Coding

Nhu-Hoang Nguyen[1,2], Tan-Phat Dang[1,2], Thanh-Dat Bui[1,2], Trong-Thuc Hoang[3], Cong-Kha Pham[3], and Huu-Thuan Huynh[1,2(✉)]

[1] University of Science, Ho Chi Minh City, Vietnam
hhthuan@hcmus.edu.vn
[2] Vietnam National University, Ho Chi Minh City, Vietnam
[3] University of Electro-Communications (UEC), Tokyo, Japan

Abstract. This paper focuses on implementing and optimizing a 2D Integer Discrete Cosine Transform (DCT) hardware accelerator to provide an efficient and flexible solution for the H.266 video compression standard. Unlike previous standards, H.266 introduces additional complexities in the 2D DCT computation, such as support for non-square blocks and high frequency removal. Our proposed design offers a versatile implementation method for the 2D Integer DCT hardware at very high speed. Our approach aims to provide a flexible and efficient solution, achieving better results than other methods. The results were tested on the Stratix 10 FPGA, utilizing 35,799 ALMs at the frequency of 353.23 MHz, and throughput reached 11,482.77 Mega pixels/s supporting video resolutions up to 16K@60fps. The data was also verified by using VTM reference software [1].

Keywords: Integer DCT · VVC · Hardware Accelerator

1 Introduction

The demand for efficient and high-quality video compression is increasing in today's digital age, especially in communication and entertainment applications. Versatile Video Coding (VVC) [2–4] was finalized in July 2020 as the latest international video encoding standard. It was developed by the Joint Video Experts Team (JVET) of the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). VVC represents a significant advancement in this field and promises data bandwidth savings and improves image quality. To implement an effective hardware architecture for the H.266 encoder, algorithms requiring computational complexity, such as Discrete Cosine Transform (DCT) [5–7], Context-Based Adaptive Binary Arithmetic Coding (CABAC) [8–10], and so on, must be efficiently designed and optimized.

There are many researches focusing on the hardware design with efficient architectures, such as a reusable architecture for various Transform Unit (TU)

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
O. Gervasi et al. (Eds.): ICCSA 2024 Workshops, LNCS 14815, pp. 110–121, 2024.
https://doi.org/10.1007/978-3-031-65154-0_7

sizes [11] or integrating forward and inverse transforms [12] and so on. However, the new features of H.266, such as non-square blocks and high-frequency removal, have not been solved yet. In this context, our research aims to provide efficient and flexible hardware for the 2D DCT. This DCT hardware architecture supports all previous video compression standards, including Advanced Video Coding (AVC) and High Efficiency Video Coding (HEVC) standards. Additionally, it accommodates new features of H.266, such as square and rectangular block sizes and high-frequency removal.

This paper is structured as follows: Sect. 2 presents related work on integer DCT implemented on hardware, serving as essential references for our research. Section 3 provides fundamental knowledge about integer DCT, including key features and the method for decomposing the kernel matrix. Our proposed architecture is reviewed in Sect. 4, covering the general architecture and detailed structures of each component. Section 5 presents the synthesis results of the proposed architecture, which was verified by version 9.0 of the VVC reference software called VVC Test Model (VTM). And compares them with those of other architectures. Finally, Sect. 6 concludes the paper.

2 Related Works

In the fields of hardware implementation of Integer DCT, there have been several prior researches aimed at optimizing performance and integration into modern video compression standards. The research [13] proposed a new hardware architecture for 2D DCT computation on Xilinx Artix-7 FPGA (Zynq-7000). Their proposal utilizes accelerated integer 2-D hardware, with 34421 LUTs and 8729 flip-flops (FF). The maximum clock frequency achieved is 289 MHz, supporting the HEVC standard and capable of processing up to 30 frames for 8K resolution. In [14], the authors presented a novel architecture structure for 2D integer DCT computation. Their architecture was described in VHDL language and synthesized on the Intel FPGA device. The hardware architecture throughput reaches processing speeds of up to 52 million pixels per second at a clock frequency of 90 MHz. The intellectual property (IP) core is implemented and verified using a system-on-chip (SoC) embedded video system. In [15], the authors introduced a hardware architecture for a 2D discrete cosine transform dedicated to the HEVC standard. This architecture supports 4K@30 frames per second with hardware cost reduction ranging from 31–64%. The hardware architecture of 2D DCT with a Compression Ratio of 5:3 was proposed in [16] with the purpose reduces computational complexity and power consumption. Specifically, a compression ratio of approximately 5:3 is used to reduce area as well as power consumption. A proposed algorithm for determining the number of DCT Output Coefficients was mentioned in [17]. This study proposes an algorithm to determine the number of DCT output coefficients such that the peak signal-to-noise ratio (PSNR) decreases by less than 0.15 dB. This algorithm can process up to 4K@288fps. These researches have contributed to the efficient implementation of Integer DCT on hardware, and they are essential references for our work.

3 Basic Knowledge

To reduce discrepancies between encoding and decoding during the 2D DCT computation and simplify hardware implementation, some researchers have applied matrix multiplication and partial butterfly implementation for DCT calculation [11,18,19]. This section outlines the essential characteristics of integer DCT and the matrix decomposition approach to enhance hardware efficiency.

3.1 Key Feature of Integer DCT

The key feature of the Integer DCT is its ability to perform lossy compression of digital images or video signals while preserving crucial visual information. Unlike the traditional DCT, which operates on floating-point numbers, the Integer DCT processes integer-valued input data. This property makes it suitable for implementation on hardware platforms, where integer arithmetic operations are more efficient than floating-point operations.

The formula (1) can generalize the Integer DCT:

$$X = C_N \times x \tag{1}$$

In which X is the coefficient after the DCT transformation, C_N is the kernel matrix, and x is the input coefficient for the DCT.

Additionally, the N-point integer DCT can be computed by utilizing $N/2$ point DCT and a vector of size $N/2$ using the butterfly method proposed in [19,20], represented by the formula (2):

$$\begin{aligned} X_{Even} &= C_{N/2} \times a \\ X_{Odd} &= M_{N/2} \times b \\ a_i &= x_i + x_{N-1-i} \quad \text{Where } 0 \le i \le N/2-1 \\ b_i &= x_i - x_{N-1-i} \quad \text{Where } 0 \le i \le N/2-1 \end{aligned} \tag{2}$$

X_{Even} and X_{Odd} denote the output coefficients of even and odd rows. $C_{N/2}$ is the left half of the even row of the matrix C_N, while $M_{N/2}$ is the matrix that has unique coefficients represented in Table 1.

Table 1. The table presents the coefficients of the DCT matrix.

Type	Unique coefficient
DCT 2	64
DCT 4	83, 36
DCT 8	89, 75, 50, 18
DCT 16	90, 87, 80, 70, 57, 43, 25, 9
DCT 32	90, 88, 85, 82, 78, 73, 67, 61, 54, 46, 38, 31, 22, 13, 4
DCT 64	91, 90, 88, 87, 86, 84, 83, 81, 79, 77, 73, 71, 69, 65, 62, 59, 56, 52, 48, 44, 41, 37, 33, 28, 24, 20, 15, 11, 7, 2

3.2 Matrix Decomposition

Each TU block size in the Integer DCT transformation has a distinct kernel matrix with corresponding coefficients listed in Table 1. These matrices share a common property: even rows are symmetric about the center, while odd rows are symmetric but flipped around the center. Additionally, larger-sized matrices' even rows contain matrices of smaller sizes. Based on these characteristics, during computation, we can decompose the matrix into smaller ones to reduce complexity and reuse architecture for smaller-sized DCTs to compute larger-sized ones, thereby optimizing hardware resources.

The kernel matrices for 4, 8, 16, and 32 points integer DCTs are provided in [19]. Applying this kernel matrix to compute the 8-point DCT, we obtain Eq. (3).

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix} = \begin{bmatrix} 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\ 89 & 75 & 50 & 18 & -18 & -50 & -75 & -89 \\ 83 & 36 & -36 & -83 & -83 & -36 & 36 & -83 \\ 75 & -18 & -89 & -50 & 50 & 89 & 18 & -75 \\ 64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 \\ 50 & -89 & 18 & 75 & -75 & -18 & -89 & -50 \\ 36 & -83 & 83 & -36 & -36 & 83 & -83 & 36 \\ 18 & -50 & 75 & -89 & 89 & -75 & 50 & -18 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \tag{3}$$

This multiplication can be decomposed into two simpler multiplications as (4a) and (4b).

$$\begin{bmatrix} X_0 \\ X_2 \\ X_4 \\ X_6 \end{bmatrix} = \begin{bmatrix} 64 & 64 & 64 & 64 \\ 83 & 36 & -36 & -83 \\ 64 & -64 & -64 & 64 \\ 36 & -83 & 83 & -36 \end{bmatrix} \times \begin{bmatrix} x_0 + x_7 \\ x_1 + x_6 \\ x_2 + x_5 \\ x_3 + x_4 \end{bmatrix} \tag{4a}$$

$$\begin{bmatrix} X_1 \\ X_3 \\ X_5 \\ X_7 \end{bmatrix} = \begin{bmatrix} 89 & 75 & 50 & 18 \\ 75 & -18 & -89 & -50 \\ 50 & -89 & 18 & 75 \\ 18 & -50 & 75 & -89 \end{bmatrix} \times \begin{bmatrix} x_0 - x_7 \\ x_1 - x_6 \\ x_2 - x_5 \\ x_3 - x_4 \end{bmatrix} \tag{4b}$$

In which the computation of X_0, X_2, X_4, and X_6 can be further decomposed into two multiplications of smaller matrices (5a) and (5b).

$$\begin{bmatrix} X_0 \\ X_4 \end{bmatrix} = \begin{bmatrix} 64 & 64 \\ 64 & -64 \end{bmatrix} \times \begin{bmatrix} (x_0 + x_7) + (x_3 + x_4) \\ (x_1 + x_6) + (x_2 + x_5) \end{bmatrix} \tag{5a}$$

$$\begin{bmatrix} X_2 \\ X_6 \end{bmatrix} = \begin{bmatrix} 83 & 36 \\ 36 & -83 \end{bmatrix} \times \begin{bmatrix} (x_0 + x_7) - (x_3 + x_4) \\ (x_1 + x_6) - (x_2 + x_5) \end{bmatrix} \tag{5b}$$

Applying this matrix decomposition approach, we can compute a 64-point integer DCT using 32, 16, 8, 4, and 2-point integer DCTs.

4 Proposed Architecture

This section introduces the proposed architecture for computing the integer DCT. It aims to reduce the number of operations by replacing multiplications with shifts and additions. Moreover, this architecture utilizes the pipeline technique to improve processing throughput.

4.1 The General Architecture

Our general architecture is illustrated in Fig. 1, the control unit oversees the overall DCT process. Specifically, when a *start* signal is triggered, the control unit decodes the *width* and *height* of the input coefficients to generate Idx value for the next steps including the DCT and the Transpose Memory (TM) to operate correctly with the input matrix size. Additionally, the control unit creates $shift$ signal for rounding and converting the output coefficients to a 16-bit size at the end of the 1D DCT calculation step. Furthermore, the control unit transmits logic "0" to the MUX to select the data stream into the DCT as the input coefficients. After 11 clock cycles, the control unit generates a *write* signal to store the DCT result of the first row in the TM. Since our architecture utilizes pipeline techniques, the result of one row is computed after each clock cycle. Therefore, after the subsequent N clock cycles, the 1D DCT computation is completed, and all the results are stored in the TM row by row. Next, the control unit sends logic "1" to the MUX to multiply the DCT value from the TM while simultaneously sending a *read* signal to the TM to retrieve the data and pass it to the 1D DCT. After 11 clock cycles, the *Datavalid* signal is activated to indicate the availability of output data. In the following N clock cycles, all output data will be retrieved until the *done* signal is activated.

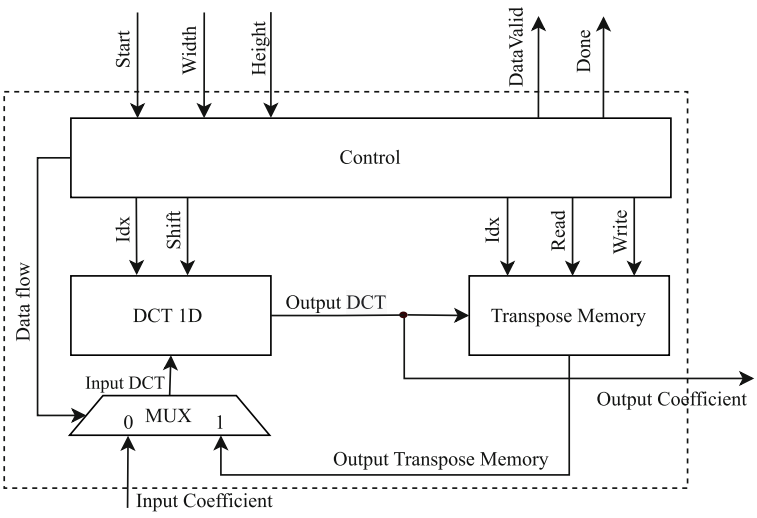


Fig. 1. The general architecture.

4.2 1D DCT Architecture

In this section, the architecture for implementing the integer DCT on hardware was introduced. This architecture consists of three main steps including Add/Sub, Odd, and Add steps, each responsible for a specific part of the computation process as shown in Fig. 2.

Add/Sub Step: This step performs addition and subtraction operations between the input coefficients following a predefined sequence [11]. As shown in Fig. 3, the resulting data from the Add/Sub step is divided into two parts: one is sent to the odd block, and the other is directed to N/2 point DCT block, which is the same 1D DCT architecture but smaller in size (N/2). For example as mentioned in Sect. 3.2, the first part is matrix b in formula 2, and this matrix is used in formula 4b. The remaining part is put into a smaller 1D DCT block to continue decomposing in the form of formulas 5a and 5b.

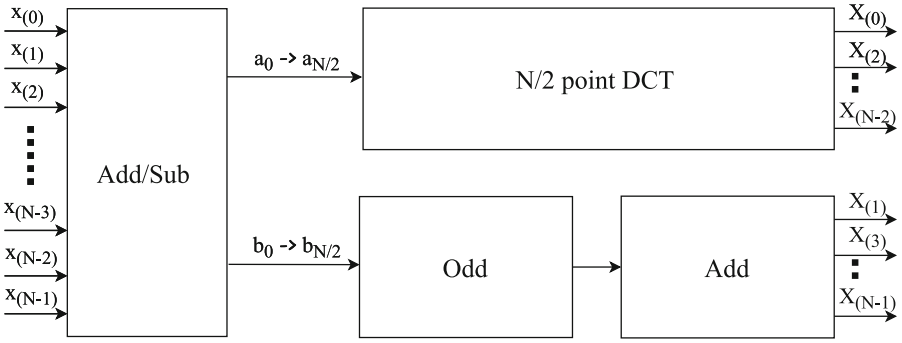


Fig. 2. 1D DCT architecture.

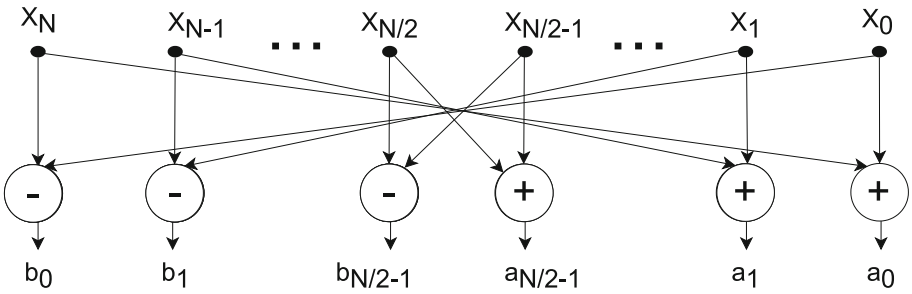


Fig. 3. Add Sub architecture.

Odd Step: This step receives the results from the Add/Sub step. Instead of performing matrix multiplication, we employ shifting and addition operations to simplify hardware complexity. Additionally, we integrate pipelining techniques during the 1D DCT calculation. To achieve this, we design an Odd architecture to ensure all outputs appear after two addition operations. Specific coefficient values in the kernel matrix are changed in sign to meet this requirement. Figure 4 illustrates our proposed architecture for the Odd block of the 16-point DCT, where 90x and 87x are replaced with −90x and −87x, respectively. This approach is also applied to all Odd blocks of varying sizes.

Add Step: The Add step conducts addition or subtraction operations between the elements of the result matrix from the Odd step in a specific sequence. Because we reversed the signs of some coefficients in the Odd step, our architecture reversed them back before performing addition along the binary tree. Additionally, we merge the subtraction operations to minimize the number of subtractions in the circuit, aiming to optimize the circuit size during synthesis. So, each output result is computed from a different Adder circuit. An example of the Add block in the 16-point DCT is depicted in Fig. 5. The result of this step is the final result of the integer DCT. By combining the Odd and Add steps, The multiplication as in formula 4b can calculated by using only shift and addition operations. Furthermore, the proposed architecture also minimizes the number of calculation cycles of this step.

Moreover, the proposed architecture includes a smaller module that performs similar computations to the overall structure but on a smaller scale. The X_{Even} in Eq. 2 was calculated by this module, and it can be decomposed into two simpler multiplications as 5a and 5b.

Pipeline Architecture: To reduce the number of clock cycles consumed during the 1D DCT calculation, the proposed architecture employs pipeline techniques. This way, our architecture can finish the 1D DCT for the first 64 points using only ten cycles (as illustrated in Fig. 6). After the subsequent 64 cycles, the entire 64 × 64 block is computed. In this pipeline architecture, the Add/Sub step takes one clock cycle, and the Odd step takes two clock cycles for all N-point integer DCT operations. The number of clock cycles for the Add step increases incrementally with the size of the kernel matrix. Additionally, our architecture includes an additional rounding stage to round and resize the output coefficients to 16 bits.

4.3 Transpose Memory Architecture

Our proposal utilizes N single-port RAMs arranged in an array to optimize memory access. By organizing the write addresses of the RAM, our architecture enables reading/writing 64 coefficients in a single cycle. Figure 7 illustrates an example of the TM block used for DCT 8×8, comprising eight separate single-port RAMs. During the writing process to TM, each row of the 1D DCT

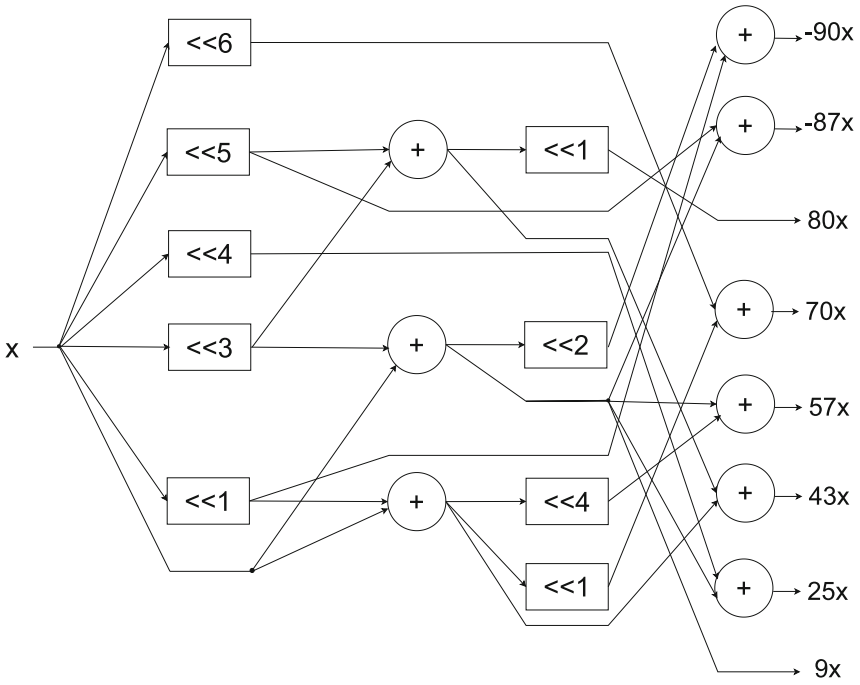


Fig. 4. The Odd architecture for the 16-point DCT.

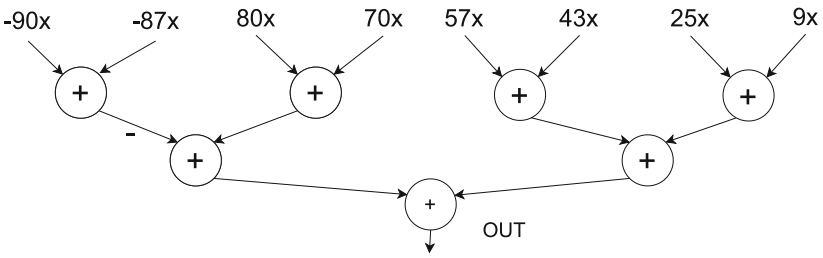


Fig. 5. An example of the Add architecture for the 16-point DCT.

result is sequentially written into TM along the forward diagonals, starting from the main diagonal. When reading from TM for the second round of 1D DCT calculation, all 8 RAMs are simultaneously read at the same address.

5 Results

In this section, our architecture was implemented and evaluated in two parts. The first part only supports a maximum size of 32×32 for easier comparison with previous studies, as those studies were applied only to the H.265 compression standard, where the DCT is applied up to a maximum block size of 32.

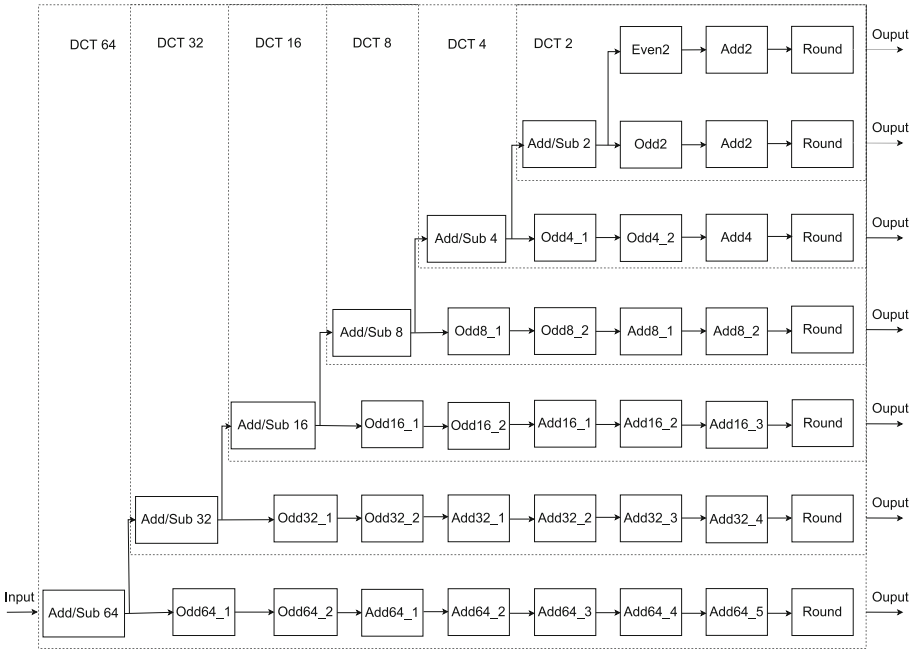


Fig. 6. Pipeline architecture.

MEM 0	Y00	Y71	Y62	Y53	Y44	Y35	Y26	Y17
MEM 1	Y10	Y01	Y72	Y63	Y54	Y45	Y36	Y27
MEM 2	Y20	Y11	Y02	Y73	Y64	Y55	Y46	Y37
MEM 3	Y30	Y21	Y12	Y03	Y74	Y65	Y56	Y47
MEM 4	Y40	Y31	Y22	Y13	Y04	Y75	Y66	Y57
MEM 5	Y50	Y41	Y32	Y23	Y14	Y05	Y76	Y67
MEM 6	Y60	Y51	Y42	Y33	Y24	Y15	Y06	Y77
MEM 7	Y70	Y61	Y52	Y43	Y34	Y25	Y16	Y07

Fig. 7. Transpose Memory Architecture for DCT 8×8.

Second one supports a size of 64×64 to accommodate all new video compression standards. All results were verified by compare with output of VTM.

5.1 The Comparison with the Same DCT Size

In this comparison, our proposed architecture only supports sizes ranging from 2 to 32 for comparison with other architectures. The Odd and Add steps are designed to minimize the number of clock cycles. This architecture also uses the pipeline technique to speed up the 1D DCT step, and the control unit

Table 2. The synthesis results on Xilinx's boards utilizing identical TU size.

	FPGA	Resources				F (Mhz)	Throughput (Mega pixels/s)
		R	LUT	Slice	DSP		
Our work	Virtex 7	27147	27723	8341	0	172	1918
[13]	Artix-7	8729	34421	N/A	N/A	289	1541
[15]	Virtex 7	N/A	5 600	N/A	128	177	361

Table 3. The synthesis results on Intel's boards utilizing identical TU size.

	FPGA	Resources (ALUT)	F (Mhz)	Throughput (Mega pixels/s)
Our work	Stratix III	25 692	316	3520
[15]	Stratix III	52 000	206	421

is used to handle non-square blocks. We conducted comparisons across Xilinx and Intel platforms, and the results show that our architecture consumes more resources but achieves higher throughput, as shown in Table 2 and Table 3. In the design [13], they achieve a higher frequency but with lower throughput because they propose utilizing a processing element (PE) architecture for computing the Odd block, where each PE block requires six clocks to produce a result. Our Odd architecture only takes two clocks to return a result. Consequently, this research has lower throughput compared to our architecture. In the design [15], they consume many clocks for computing the 1D integer DCT, specifically 256 clocks for the 32×32 block. In contrast, our design employs pipeline techniques, requiring only 42 clocks to complete the 1D integer DCT computation for the 32×32 block. Thus, it achieves relatively limited throughput compared to our design.

5.2 The 64×64 Block Results

Our proposed architecture applies pipelining techniques, so it has a larger hardware resource when implemented. Furthermore, our design utilizes fewer clock cycles to compute the 2D DCT, requiring only 126 clock cycles for a 64×64 block. The implementation results on various chips are demonstrated in Table 4.

Table 4. The experimentation results of our proposed architecture at size 64×64.

FPGA	Resources	F (Mhz)	Throughput (Mega pixels/s)
Stratix III	55 061 ALUT	260	8450
Cyclone IV	64 828 LE	117	3815
Stratix 10	35 799 ALM	352	11482
Artix-7	47472 R, 59573 LUT, 17102 Slice	107	3505

When implemented on an Intel Stratix 10 chip, the maximum frequency achieved is 352 MHz, with a throughput of 11482 Mega pixels/s, supporting video resolutions up to 16k@60fps.

6 Conclusion

In conclusion, the 2D integer DCT Hardware Accelerator was implemented and optimized on an FPGA, focusing on supporting the H.266 video compression standard. Our proposed design has demonstrated efficiency and flexibility in handling the complexities introduced by H.266, such as non-square blocks and high-frequency removal. The design was tested on Stratix 10 FPGA at a throughput of 11,482.77 Mega pixels/s and ready for 16K@60fps. Our design also supports previous video compression standards and accommodates the advanced features of H.266, making it a versatile solution for various video compression applications.

Acknowledgments. This research is funded by University of Science, VNU-HCM under grant number ĐT-VT 2023-06.

References

1. The VTM reference software. https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM. Accessed 25 Mar 2024
2. Bross, B., et al.: Overview of the versatile video coding (VVC) standard and its applications. IEEE Trans. Circuits Syst. Video Technol. **31**(10), 3736–3764 (2021). https://doi.org/10.1109/TCSVT.2021.3101953
3. Wieckowski, A., Brandenburg, J., Bross, B., Marpe, D.: VVC search space analysis including an open, optimized implementation. IEEE Trans. Consum. Electron. **68**(2), 127–138 (2022). https://doi.org/10.1109/TCE.2022.3148813
4. Bossen, F., Sühring, K., Wieckowski, A., Liu, S.: VVC complexity and software implementation analysis. IEEE Trans. Circuits Syst. Video Technol. **31**(10), 3765–3778 (2021). https://doi.org/10.1109/TCSVT.2021.3072204
5. Zhao, X., et al.: Transform coding in the VVC standard. IEEE Trans. Circuits Syst. Video Technol. **31**(10), 3878–3890 (2021). https://doi.org/10.1109/TCSVT.2021.3087706
6. Mukherjee, D., Mukhopadhyay, S.: hardware efficient architecture for 2D DCT and IDCT using taylor-series expansion of trigonometric functions. IEEE Trans. Circuits Syst. Video Technol. **30**(8), 2723–2735 (2020). https://doi.org/10.1109/TCSVT.2019.2928045
7. Masera, M., Masera, G., Martina, M.: An area-efficient variable-size fixed-point DCT architecture for HEVC encoding. IEEE Trans. Circuits Syst. Video Technol. **30**(1), 232–242 (2020). https://doi.org/10.1109/TCSVT.2018.2886736
8. Ramos, F.L.L., Saggiorato, A.V.P., Zatt, B., Porto, M., Bampi, S.: Residual syntax elements analysis and design targeting high-throughput HEVC CABAC. IEEE Trans. Circuits Syst. I Regul. Pap. **67**(2), 475–488 (2020). https://doi.org/10.1109/TCSI.2019.2932891

9. Zhang, Y., Lu, C.: A highly parallel hardware architecture of table-based CABAC bit rate estimator in an HEVC intra encoder. IEEE Trans. Circuits Syst. Video Technol. **29**(5), 1544–1558 (2019). https://doi.org/10.1109/TCSVT.2018.2830126
10. Karwowski, D.: Precise estimation of probabilities in CABAC using the cauchy optimization method. IEEE Access **8**, 32088–32099 (2020). https://doi.org/10.1109/ACCESS.2020.2973549
11. Meher, P.K., Park, S.Y., Mohanty, B.K., Lim, K.S., Yeo, C.: Efficient integer DCT architectures for hevc. IEEE Trans. Circuits Syst. Video Technol. **24**(1), 168–178 (2014). https://doi.org/10.1109/TCSVT.2013.2276862
12. Budagavi, M., Sze, V.: Unified forward+inverse transform architecture for hevc. In: 2012 19th IEEE International Conference on Image Processing, pp. 209–212 (2012). https://doi.org/10.1109/ICIP.2012.6466832
13. Ansari, A.E., Mansouri, A., Ahaitouf, A.: An efficient VLSI architecture design for integer DCT in hevc standard. In: 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA), pp. 1-5 (2016). https://doi.org/10.1109/AICCSA.2016.7945747
14. Loukil, H., Masmoudi, N.: A novel architecture design for VLSI implementation of integer DCT in HEVC standard. Multimed. Tools Appl. **79**, 23977–23993 (2020). https://doi.org/api.semanticscholar.org/CorpusID:219710153
15. Chen, M., Zhang, Y., Lu, C.: Efficient architecture of variable size hevc 2d- dct for fpga platforms. AEU - Int. J. Electron. Commun. **73**, 1–8 (2017). https://doi.org/10.1016/j.aeue.2016.12.024
16. Akman, A., Cekli, S.: Efficient 2d DCT architecture based on approximate compressors for image compression with HEVC intra-prediction. J. Real-Time Image Proc. **20**(2), 22 (2023)
17. Singhadia, A., Mamillapalli, M., Chakrabarti, I.: Hardware-efficient 2D-DCT/IDCT architecture for portable HEVC-compliant devices. IEEE Trans. Consum. Electron. **66**(3), 203–212 (2020)
18. Abdelrasoul, M., Sayed, M.S., Goulart, V.: Scalable integer DCT architecture for HEVC encoder. In: 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 314–318 (2016). https://doi.org/10.1109/ISVLSI.2016.98
19. Budagavi, M., Fuldseth, A., Bjøntegaard, G., Sze, V., Sadafale, M.: Core transform design in the high efficiency video coding (HEVC) standard. IEEE J. Sel. Top. Signal Process. **7**(6), 1029–1041 (2013). https://doi.org/10.1109/JSTSP.2013.2270429
20. Chen, W.-H., Smith, C., Fralick, S.: A fast computational algorithm for the discrete cosine transform. IEEE Trans. Commun. **25**(9), 1004–1009 (1977). https://doi.org/10.1109/TCOM.1977.1093941