A Pipelined 2D Transform Architecture Supporting Mixed Block Sizes for the VVC Standard

Yibo Fan, Yixuan Zeng, Heming Sun, Jiro Katto, *Member, IEEE*, and Xiaoyang Zeng, *Member, IEEE*

***Abstract*— For the next-generation video coding standard Versatile Video Coding (VVC), several new contributions have been proposed to improve the coding efficiency, especially in the transformation operations. This paper proposes a unified 32 x 32 block-based transform architecture for the VVC standard that enables 2D Discrete Sine Transform-VII (DST-VII) and Discrete Cosine Transform-VIII (DCT-VIII) of all sizes. It mainly gives three contributions: 1) The N-Dimensional Reduced Adder Graph (RAG-n) algorithm is adopted to design the minimal adder-oriented computational units. 2) The storage of the asymmetric transform units can be realized in the dual-port SRAM-based transpose memory. 3) The pipelined 2D transformations of mixed block sizes are achieved with the throughput rate of 32 samples per cycle. The synthesis results indicate that this architecture can reduce area by up to 73.1% compared with other state-of-the-art works. Moreover, power saving ranging from 4.9% to 9.9% can be achieved. Regarding the transpose memory, at least 21.9% of the area can be saved by using SRAM.**

***Index Terms*— Versatile Video Coding, transform, pipeline, DST-VII, DCT-VIII.**

I. Introduction

VIDEO compression is very important in the reduction of the storage space of video data. The International Telecommunication Union (ITU) and the International Organization for Standardization (ISO) are jointly developing a new international video coding standard called Versatile Video Coding (VVC) [1]. VVC is expected to achieve 50% coding efficiency gain over the current video coding standard High Efficiency Video Coding (HEVC) [2] at the same level of video quality. It is worth noting that these superior coding performances are obtained at the expense of high computational complexity. In particular, one of the most frequently performed modules is the variable size transform computation.

VVC makes three changes in transformation operations: 1) Besides retaining Discrete Cosine Transform-II (DCT-II) adopted by HEVC, Discrete Sine Transform-VII (DST-VII) and Discrete Cosine Transform-VIII (DCT-VIII) are added as the new transform types. 2) Using different transform types in two (horizontal and vertical) transform directions is supported. 3) A more flexible block partitioning method is employed, which supports both asymmetric block sizes and symmetric block sizes.

Several low-cost hardware architectures have been developed for HEVC DCT-II in the past few years. For the trade-off between hardware cost and work efficiency, Shen et al. [3] proposed a mixture of multipliers and addition/shift operations to realize matrix multiplication. A unified architecture for all of the transform sizes was proposed by Meher et al. [4] using Chen's algorithm [5]. A fast algorithm for eight-point integer transform was introduced by Sun et al. [6] to reduce the computational complexity. A cyclic memory organization and a reordered parallel-in serial-out scheme [7] were employed to reduce the hardware cost. Zhang and Lu [8] presented a fully parallel hardware architecture for an HEVC encoder, including transformation operations. By only calculating several pre-determined low-frequency coefficients of transform units (TUs), Kalali et al. [9] significantly reduced the computational complexity of DCT-II. A novel truncation scheme [10] was proposed to implement the approximated transform design. Pastuszak and Abramowski [11] introduced a computationally scalable algorithm for an intra encoder.

However, because of the differences of the transform sizes and types, the previous methods cannot be directly used in the VVC transforms. Several hardware implementations [12]–[16] for DST-VII/ DCT-VIII have been proposed. The 1D transforms of only small sizes (four-point and eight-point) were implemented by Kammoun et al. [12] and Ben Jdidia et al. [13], respectively. Mert et al. [14] realized two different types of high-performance 2D transform hardware for 4×4 and 8×8 TUs. Kammoun et al. [15] and Garrido et al. [16] proposed a unified 2D transform architecture at the expense of many hardware resources. These designs were both FPGA-based designs, using either FPGA multipliers or DSPs. It is difficult to compare them with the ASIC-based designs.

Regarding the transpose memory, some effective storage schemes were described in the literature. The transpose memory proposed by Mert et al. [14] consists of eight SRAMs, each of which is 8×32 bits, which is enough to realize the transform up to the 8×8 block size. Chen et al. [17] and Sjövall et al. [18] exploited the same storage space to support the 32×32 transform. The only difference is the word width. Each word unit can store 4 points ($4 \times 16 = 64$ bits) and 1 point (16 bits) respectively for [17] and [18]. An advanced diagonal storage strategy which enables data to be obtained parallelly was described in [19].

Manuscript received May 13, 2019; revised July 21, 2019; accepted July 31, 2019. Date of publication August 12, 2019; date of current version September 3, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61674041, in part by the Alibaba Innovative Research (AIR) Program, in part by the IBM Faculty Award, in part by the Innovation Program of Shanghai Municipal Education Commission, and in part by the Pioneering Project of Academy for Engineering and Technology and Fudan-CIOMP Joint Fund. This article was recommended by Associate Editor C.-T. Huang. *(Corresponding author: Heming Sun.)*

Y. Fan, Y. Zeng, and X. Zeng are with the State Key Laboratory of ASIC and System, Fudan University, Shanghai 200433, China (e-mail: fanyibo@fudan.edu.cn; 18210860015@fudan.edu.cn; xyzeng@fudan.edu.cn).

H. Sun is with the Waseda Research Institute for Science and Engineering, Tokyo 169-8555, Japan (e-mail: hemingsun@aoni.waseda.jp).

J. Katto is with the Graduate School of Fundamental Science and Engineering, Waseda University, Tokyo 169-8555, Japan (e-mail: katto@waseda.jp).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2019.2934752

TABLE I
BASIS FUNCTIONS OF THE TRANSFORMATIONS IN VVC

Transform Type	Basis function Ti(j), i, j = 0, 1, …, N−1
DCT-II	$T_i(j) = \omega_0 \cdot \sqrt{\frac{2}{N}} \cdot \cos(\frac{\pi \cdot i \cdot (2j+1)}{2N})$
DCT-VIII	$T_i(j) = \sqrt{\frac{4}{2N+1}} \cdot \cos(\frac{\pi \cdot (2i+1) \cdot (2j+1)}{4N+2})$
DST-VII	$T_i(j) = \sqrt{\frac{4}{2N+1}} \cdot \sin(\frac{\pi \cdot (2i+1) \cdot (j+1)}{2N+1})$

Source: Working Draft 4 of Versatile Video Coding [1].

This paper presents a high-performance hardware implementation for 2D DST-VII/DCT-VIII operations in VVC. The main contributions of this design are as follows. 1) **Minimal adder-oriented logical computational units:** To minimize the number of adders as much as possible, the N-Dimensional Reduced Adder Graph (RAG-n) algorithm [20] is adopted to design the logical computation parts of VVC transforms. 2) **Transpose memory supporting the storage of asymmetric blocks:** Dual-port SRAM is employed as transpose memory, and the storage of various asymmetric blocks can be realized by a diagonal scheme. 3) **A** 32×32 **block-based pipeline operation:** The 2D transformations of mixed block sizes can be achieved in a pipeline, and 32 transformed coefficients can be calculated each cycle after a certain initial delay.

The rest of this paper is organized as follows. In Section II, the basic principles and algorithms are introduced. The proposed pipelined 2D transform architecture, including transpose memory, is described in detail in Section III. Section IV gives the experimental results and comparisons. Section V concludes this paper.

II. BASIC PRINCIPLES

A. The Transformation Operations in VVC

Since the transforms could eliminate the correlations among prediction residuals and concentrate the residuals' energy, it is widely used in various image and video coding standards. HEVC uses DCT-II as the main transform function, and DST-VII is used for intra coding of a 4×4 luminance component. Moreover, both the horizontal transformation and vertical transformation use the same transform type, and the TUs are symmetric blocks, containing 4×4, 8×8, 16×16 and 32×32 block sizes [2].

For the coding efficiency gain, VVC uses three transform kernels: DCT-II, DST-VII and DCT-VIII. Table I presents the different basis functions of the selected DCT/DST type. Meanwhile, the horizontal and vertical transformations may adopt different transform types. In the reference software (VTM4.0) [21], two additional flags are added for signaling the horizontal and vertical transform types. Additionally, a more flexible partitioning mode is adopted. This partitioning scheme ensures that asymmetric transform blocks (such as 16×4 and 32×8) are supported.

There is no doubt that these new strategies have significantly improved the coding performance, but they bring about a large amount of the computational complexity. In hardware design, the resource consumption is closely related to the computational complexity. Thus, it is desired to design a low-cost and high-performance hardware structure for transformation operations.

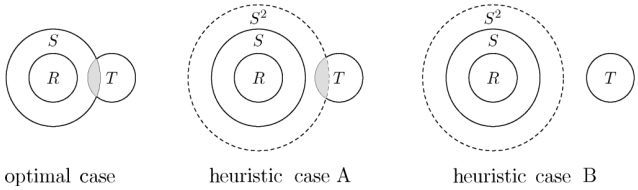


Fig. 1. Three cases considered by the RAG-n heuristic.

B. The RAG-n Algorithm

In some previous studies, Multiplierless Multiple Constant Multiplication (MCM) [20] was often used to design multiplier blocks by only adopting additions, subtractions and shifts. Due to the low hardware cost of shift operations, how to minimize the number of adders and subtractors is an important goal of MCM. RAG-n is a graph-based MCM algorithm that is dedicated to reducing the number of adders and subtractors. It requires a precomputed table of optimal Single Constant Multiplication (SCM) decompositions that are obtained by exhaustive search. First, we give some relevant definitions.

Fundamentals: A multiplier block implements the parallel multiplication by a given set of constants, which we call fundamentals.

A-operation: It performs a single addition or subtraction and an arbitrary number of shifts.

A-distance: The minimum number of extra A-operations required to obtain a fundamental.

T: The target set of constants.

R: A set of original fundamentals (the ready set).

S: A set of fundamentals that can be obtained by only one A-operation with the given initial constants.

S^2: A set of fundamentals that can be obtained by two A-operations with the given initial constants.

The target A-operation in RAG-n is A_{odd}, which means that all of the targets T are first right shifted to become odd. The RAG-n heuristic considers three different cases, graphically illustrated in Fig. 1 and discussed thereafter.

1) **Optimal case**. If $T \cap S \neq \phi$, then there is a target in the successor set, and it is synthesized. If the entire set T is synthesized this way, then the solution is optimal, since it is impossible to use less than one A-operation for each odd target. Thus, this case is referred to as optimal.
2) **Heuristic case A**. If $T \cap S = \phi$ and $T \cap S^2 \neq \phi$, then there is a target at an A-distance of 2 from R. This target is synthesized along with the distance-1 intermediate fundamental.
3) **Heuristic case B**. If no distance-1 or distance-2 targets are available, then RAG-n synthesizes the target of least complexity using the precomputed optimal SCM table. In this case, three or more constants are synthesized.

If all of the coefficients can be synthesized in the optimal part, we can ensure that the number of adders is the least. Otherwise, the remaining coefficients are synthesized in the heuristic part, and the results are not guaranteed to be optimal.

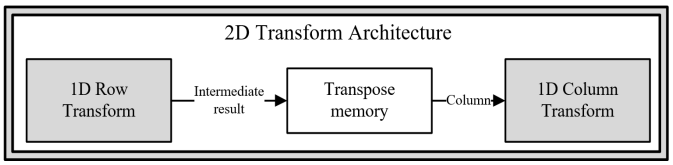


Fig. 2. Top-level module of the 2D transform (unfolded architecture).

III. Proposed Pipelined 2D Transform Architecture

The top-level structure for the pipelined 2D transform is illustrated in Fig. 2. It mainly consists of three modules: 1D row transform, 1D column transform and the transpose memory. These two 1D transform modules have the same structure, only with the different bit-width parameters. (The bit-width parameter is equivalent to the input data bit-width.) We assume that the bit-width parameters for the row transform and column transform are 9 and 16, respectively, as defined in the video coding standard. These three modules will be discussed in detail in our following paper.

A. 1D Transform

As we know, for all of the transform types, the 1D transform can be computed as follows:

$$Y = T_N \cdot X^T \tag{1}$$

In the expression above, X is a 1 × N matrix with a row of the input frame, T_N is the N × N integer transform coefficient matrix, and Y is an N × 1 column matrix with the result of the 1D transform. The 4 × 4 integer transform coefficient matrices are given below by (2) and (3) [1].

$$DST-VII_{4x4} = \begin{pmatrix} 29 & 55 & 74 & 84 \\ 74 & 74 & 0 & -74 \\ 84 & -29 & -74 & 55 \\ 55 & -84 & 74 & -29 \end{pmatrix} \tag{2}$$

$$DCT-VIII_{4x4} = \begin{pmatrix} 84 & 74 & 55 & 29 \\ 74 & 0 & -74 & -74 \\ 55 & -74 & -29 & 84 \\ 29 & -74 & 84 & -55 \end{pmatrix} \tag{3}$$

From the above DST-VII $_{4\times4}$ transform matrix (2), we can observe a feature that the data in the fourth column can be obtained by adding the data in the first and second columns, except for the second row. Similar features are also found in the 16 × 16 and 32 × 32 transform matrices. VTM4.0 has exploited this feature to simplify the computation and reduce the number of multiplication operations. For example, the first output can be obtained by (4) without the calculation of src3·84, where src0, src1, src2 and src3 are the inputs of the 1D four-point DST-VII transform module, and dst0 is the first output.

$$\text{dst0} = 29 \cdot [\text{src0} + \text{src3}] + 55 \cdot [\text{src1} + \text{src3}] + 74 \cdot \text{src2} \tag{4}$$

Moreover, another feature could be found that the first, third and fourth columns have the same coefficients only in different order and signs. In addition, the transform matrices of other sizes have such a feature. Therefore, we can design the Shift-Addition Units (SAUs) as the multiplier blocks which are used several times parallelly. To facilitate and unify the SAU designs, this paper does not employ the simplification strategy mentioned above. The SAUs are designed by employing the previously mentioned RAG-n algorithm, and SAU designs for various sizes are depicted in Fig. 3. Taking size 4 as an example, an SAU can realize four outputs ($29x$, $55x$, $74x$ and $84x$) with one input x as indicated in Fig. 3 (a). It only consumes six additions and six shifts. Table II presents the number of additions and shifts required for the SAU designs of various sizes. Through the RAG-n algorithm, we can reduce the number of adders and subtractors as much as possible.

TABLE II
The Number of Additions and Shifts for SAU

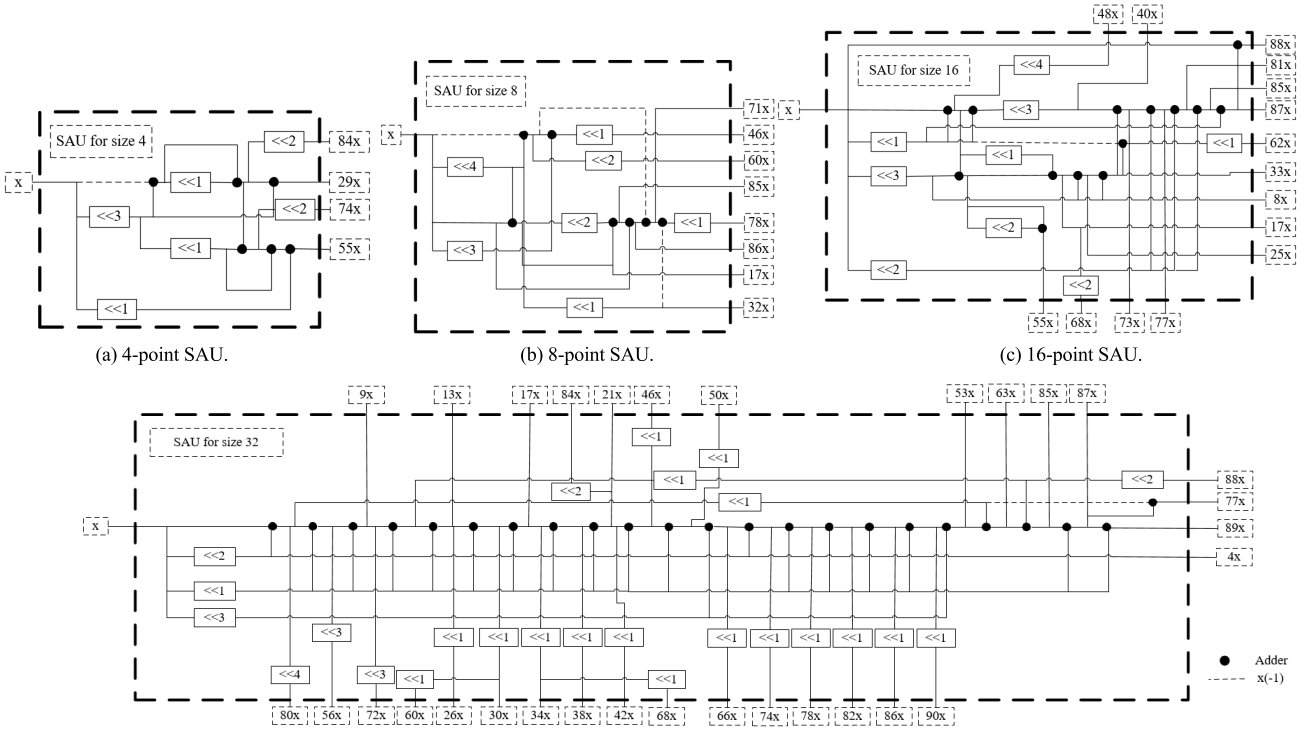
N-point SAU	The number of Additions	The number of Shifts
N=4	6	6
N=8	7	7
N=16	14	9
N=32	22	25

Additionally, we find that compared with the DST-VII matrix (2), DCT-VIII one (3) has the same coefficients but in inverse order for each row. Then, with only inversing the inputs order and assigning the appropriate outputs signs, we can easily benefit from DST-VII architecture in implementation of the DCT-VIII transform type with no additional computational complexity. However, because of the independence between the DST-VII/DCT-VIII transform matrices for different sizes, different structures are employed on the basis of transform sizes. Fig. 4 presents the generic structure of 1D DST-VII and DCT-VIII implementation for size N, where N = 4, 8, 16, 32. An N-point transform module consists of a mux, a set of SAUs (N SAUs) and an adder tree. To switch between DCT-VIII and DST-VII, we use a mux to choose whether the inputs are in a sequential order or a reversed order. Besides, the outputs are slightly different for different transform types. We control them by a **select** signal. When the **select** signal is equal to 0, we adopt DST-VII, and the inputs are sequential to the mux to calculate the outputs Y_0, Y_1, …, Y_N. When the **select** signal is equal to 1, DCT-VIII is employed, and the inputs are reversed to the mux to get the outputs Y_0, $-Y_1$, …, Y_{N-1}, $-Y_N$. A set of SAUs are employed as the core computing units. For an N-point transform module, N SAUs are required, and the SAU type depends on the value of N. For example, a four-point transform module needs four 4-point SAUs. When calculating the outputs, we only need to select the corresponding results from SAU outputs and use the appropriate signs in the adder tree.

Actually, some other papers used the pixel parallelism of 32 pixels per cycle, such as [4], [9], [10] and [11]. Therefore, we employ the same parallelism in the proposed architecture. To meet the throughput rate of 32 samples per cycle and support the transformations of various sizes, many hardware resources are consumed. A 1D transform module consists of eight 4-point transform modules, four 8-point transform modules, two 16-point transform modules and a 32-point transform module (4×8 4-point SAUs + 8×4 8-point SAUs + 16 × 2 16-point SAUs + 32 × 1 32-point SAUs).

B. Transpose Memory

As we know, the register array is not area efficient for large-size transpose memory; it is also not power efficient. Compared with the register array, it is a more cost-effective



(a) 4-point SAU. (b) 8-point SAU. (c) 16-point SAU.

(d) 32-point SAU.

Fig. 3. Shift-Addition Units (SAU) designs for various transform sizes.

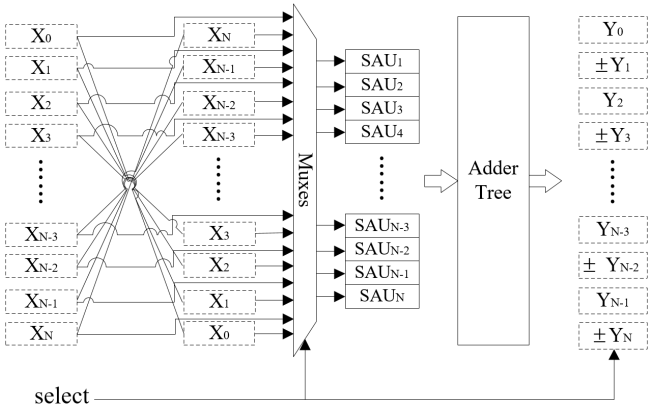


Fig. 4. Generic 1D DST-VII/DCT-VIII architecture for size N (Different SAU designs are adopted according to the value of N).

approach to implement large-size transpose memory using SRAM. To implement the pipelined architecture, we need to read and write data simultaneously. Therefore, we employ the dual-port SRAM. However, the disadvantage of SRAM is that it can only be accessed in different banks. To get data parallelly, the data of each row or column must be stored in different banks. By this, data collisions could be avoided effectively.

To achieve the throughput rate of 32 samples per cycle, the transpose memory is divided into 32 banks. Meanwhile, this paper uses a diagonal scheme to get data parallelly. The intermediate results of row transformation are written into different SRAM banks and read out in a diagonal direction for column transformation. There will be some shift operations for realizing this diagonal strategy, and some similar schemes were reported in [3], [6], [7] and [14]. When the 4×32 size is taken as an example, the storage scheme is illustrated in Fig. 5. When the transformation of one row (32 samples) is done, the intermediate results are stored in one row of the transpose memory within 32 different banks. After four clock cycles (read/write throughput rate: 32 samples per cycle), the intermediate results of the 4×32 block are all stored in the transpose memory. To ensure that each column is in the diagonal direction, the data of the second row, the third row and the fourth row are move to the right by one, two and three word units, respectively, as depicted in Fig. 5.

The depth of SRAM is set to 64, and the word width is 16 bits. (assuming the width of an intermediate result is 16 bit, and one intermediate result is stored in one word unit.) All sub-blocks in the 32×32 range can be stored with the depth 32. To implement the pipelined architecture based on the 32×32 block range, we set the depth to 64. In each **Tr** operation, we do **WR** and **RD** simultaneously. (The definitions of **Tr**, **WR** and **RD** are given below). By this interlaced reading and writing scheme, the row transformation and column transformation can be performed parallelly and the pipelined transform architecture could be realized.

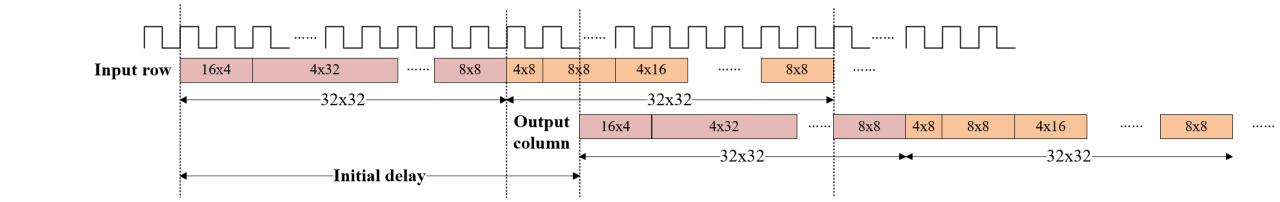
1) **Tr**: The row/column transformations of all sub-blocks in the 32×32 block range.
2) **WR**: The intermediate results of the current 32×32 block range are written into the transpose memory within the address from 0 to 31 (from 32 to 63).
3) **RD**: The intermediate results of the last 32×32 block range are read out from the transpose memory within the address from 32 to 63 (from 0 to 31).

C. A Mixture of Different Transform Sizes in a Pipeline

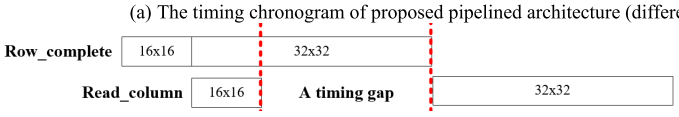
To implement this pipelined architecture supporting mixed block sizes, we consider that the maximum size of a TU is

bank0	bank1	bank2	bank3	bank4	bank5	bank6	bank7	bank8	bank9	bank10	bank11	bank12	bank13	bank14	bank15	bank16	bank17	bank18	bank19	bank20	bank21	bank22	bank23	bank24	bank25	bank26	bank27	bank28	bank29	bank30	bank31
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
32	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
31	32	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
30	31	32	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

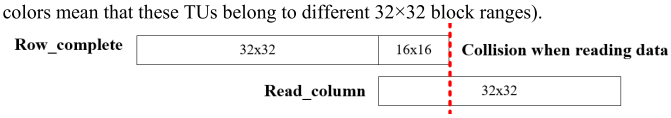
Fig. 5. Storage scheme in the case of 4×32 block (The numbers (1–32) represent that the column of the TU in which the intermediate results are. According to the different numbers and colors, it can be easily seen that each column in the 4×32 TU is stored diagonally in the transpose memory).



(a) The timing chronogram of proposed pipelined architecture (different colors mean that these TUs belong to different 32×32 block ranges).



(b) Case 1: A timing gap.



(c) Case 2: A reading data collision.

Fig. 6. Transformations of mixed sizes in the pipelined architecture (the transform sizes are indicated in the blocks).

32×32 [1]. No matter how the TU is partitioned, each sub-block will be within the 32×32 block range. Therefore, we propose a 32×32 block-based pipelined architecture. Within this range, whether it contains a 32×32 TU, four 16×16 TUs, or 32 4×8 TUs, or other combinations, we first do row transform. When the row transformations of all of the sub-blocks in this range are completed, we start reading columns out from the transpose memory for column transformations. Meanwhile, all sub-blocks in the next 32×32 block range start row transform.

All of the previous studies use either the folded architecture or the unfolded architecture (1D row transform + transpose memory + 1D column transform) as presented in Fig. 2 to do the 2D transform [4]. There is no doubt that it is impossible to get a pipelined 2D transform in the folded architecture. As for the unfolded architecture, in some previous papers, such as [4], [6], [7] and [14], the pipeline operations were realized only when the input blocks had the same size. However, these papers did not explain for such case that the input blocks have different sizes (for example, the current block is 16×16, and the next block is 32×32). This causes that the second transformation either waits for data or gets a data reading collision as illustrated in Fig. 6(b) and Fig. 6(c). A perfect pipeline cannot be found. It is obvious that the input blocks have different sizes in the decoding process. Therefore, this paper proposes a mixed-sizes-oriented design. For a perfect pipeline, all sub-blocks (which have different block sizes) in the 32×32 range do not start the column transformations until their row transformations are all completed. More delays are introduced at the beginning, but after the initial delay, 32 transformed coefficients can be calculated each clock cycle, and the 2D transformation operations for all sub-blocks in a 32×32 block range can be completed every 32 clock cycles, as illustrated in Fig. 6(a).

For this design, the initial delays are 34 clock cycles: 32 cycles for inputting the 1,024 samples (as the throughput rate is set to 32 samples per clock cycle, 32 cycles are required to get all data in the 32×32 block range), 1 cycle for capturing the intermediate results and 1 cycle for storing them in the transpose memory. After 34 cycles, the column transformations are turned on, and the results of column transformations are output in pipeline with the throughput rate of 32 outputs per clock cycle.

D. Unified Transform Architecture for Three Transform Kernels

As described in [1], VVC adopted three transform kernels (DCT-II, DCT-VIII and DST-VII). Because of the differences between their basic functions and the differences in the transform sizes (DCT-II: 2–64, DST-VII/DCT-VIII: 4–32), it is difficult to design a unified architecture. However, the proposed 1D transform architecture can be easily extended to a unified one that contains the three transform types with only an additional DCT-II engine. We can use the existing architectures, such as [3], [4] and [7], as the DCT-II engine. This unified architecture is depicted in Fig. 7, and the right side depicts our design. Thirty-two points enter two engines for the transformation calculation, and the outputs are selected by the mux with a **sel** signal.

IV. Experimental Result

We implemented the proposed designs in Verilog HDL. They were synthesized with Design Compiler using a TSMC 65 nm cell library at the clock frequency of 250 MHz. A fair comparison with other studies in the literature is quite difficult. Most studies focus on HEVC DCT-II, and there are few ASIC-based designs for the VVC transforms. For comparison, Table III lists the key parameters of state-of-the-art ASIC-based works, including the HEVC-related works ([3], [4], [7], [8]–[11]) and VVC-related literature [14]. A concept of normalized area (NA) is introduced to make a fair comparison, and NA is defined by the following equation.

$$NA = \frac{Gate}{MaxThroughput} = \frac{Gate}{MaxFreq * Tp} \tag{5}$$

TABLE III
COMPARISON OF DIFFERENT 2D TRANSFORM HARDWARE DESIGNS BASED ON ASIC

Design	[3]	[4]	[7]	[14]	Proposed SAU-based[2)]	Proposed Multiplication-based[3)]
Standard	HEVC	HEVC	HEVC	JEM[1)]	VVC(VTM4.0)	
Technology	ASIC 130nm	ASIC 90nm	ASIC 90nm	ASIC 90nm	ASIC 65nm	
Maximum Frequency (MHz)	350	187	312	345	250	
Pixel Parallelism (pixels/cycle)	4	32	4	8	32	
Gate Count (K)	109.2×2	347	63.8×2	136.4	496.4	560.4
NA	156.00	57.98	102.24	49.42	**62.05**	70.05
Power Consumption (mw)	-	67.6	-	-	62.6	69.5
Transform Unit (M×N)	M = N = 4, 8, 16, 32.	M = N = 4, 8, 16, 32.	M = N = 4, 8, 16, 32.	M = N = 4, 8.	M = 4, 8, 16, 32; N = 4, 8, 16, 32.	
Transform Type	DCT-II	DCT-II	DCT-II	DCT-II, DCT-V, DCT-VIII, DST-I,DST-VII	DST-VII, DCT-VIII	

Design	[8]	[9]	[10]	[11]	Proposed SAU-based	Proposed Multiplication-based
Standard	HEVC	HEVC	HEVC	HEVC	VVC(VTM4.0)	
Technology	ASIC 90nm	ASIC 90nm	ASIC 90nm	ASIC 90nm	ASIC 65nm	
Maximum Frequency (MHz)	320	130	187	200	250	
Pixel Parallelism (pixels/cycle)	8–16	16–32	32	8–32	32	
Gate Count (K)	426.3	197	102×2	368.9	496.4	560.4
NA	83.26–166.5	94.7	34.09	57.64–230.56	**62.05**	70.05
Power Consumption (mw)	-	65.8	-	-	62.6	69.5
Transform Unit (M×N)	M = N = 4, 8, 16, 32.	M =N = 4, 8, 16, 32.	M = N = 32.	M = N = 4, 8, 16, 32.	M=4, 8, 16, 32; N=4, 8, 16, 32.	
Transform Type	DCT-II	DCT-II	DCT-II	DCT-II	DST-VII, DCT-VIII	

1) Note: JEM is an over-stage reference software from HEVC to VVC. Many effective algorithms are included in JEM. It is different from the current H.266/VVC.

2) SAU-based design: SAUs are designed as multiplier blocks to realize the matrix multiplications. 3) Multiplication-based design: It uses multipliers directly.

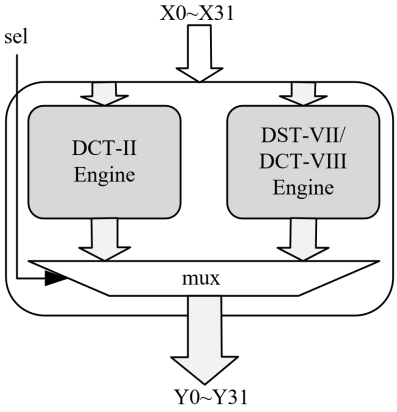


Fig. 7. Unified 1D transform architecture for VVC standard.

where MaxFreq is the maximum frequency of the design, and Tp is the throughput within each cycle, which is equal to the pixel parallelism. Gate is the gate count of the logical calculation part. It can be seen from Table III that compared with [3], [7], [8], [9] and [11], our SAU-based design has obvious advantages. We present a unified 2D transform architecture that can realize DST-VII/DCT-VIII for various asymmetric block sizes with the high throughput of 32 pixels per cycle. In term of NA, our SAU-based design could save 11.4% area than the multiplication-based one. Particularly, up to 73.1% area can be reduced compared with [11]. However, the NA is slightly larger than those of [4] and [14]. This is because Meher et al. [4] used Chen's decomposition algorithm [5] to get a unified 2D DCT-II transform architecture to reduce hardware resources. However, currently, no effective decomposition algorithms have been found for DCT-VIII and DST-VII. Although [14] had a smaller NA, the 2D transform could be achieved only up to the 8×8 block size. An approximated novel truncation scheme was adopted by Sun et al. [10], which made it possess the smallest NA with some loss in the coding performance. Regarding the power consumption, the SAU-based design can save energy by 7.4%, 4.9% and 9.9%, respectively, compared with [4], [9] and the multiplication-based design.

Additionally, considering that most of the existing designs for VVC transformation operations are based on the FPAG platform, we summarize some advanced work in Table IV. References [14]–[16] proposed some efficient architectures for five transform types. The 2D transformations for 4×4 and 8×8 TUs were performed in [14]. A unified architecture considering all asymmetric 2D block size combinations was proposed by Kammoun et al. [15]. A deeply pipelined high-performance architecture [16] was presented to implement the five transforms for symmetric blocks. For DCT-II, Chen et al. [17] proposed a methodology that can efficiently proceed 2D-DCT computation to fit internal components and characteristics of FPGA resources. Four algorithm adaptations and a fully parallel hardware architecture [8] were introduced for an HEVC intra encoder that includes the transform module.

The comparisons for the transpose memory designs are given in Table V. We can see that the area is reduced by 21.9% and 46.9%, respectively, by using SRAM, compared with that of [3] and the register array. To get a perfect pipeline, the transpose memory is divided into two areas (each of which can store 32×32 data. Read and write operations alternate

TABLE IV
DIFFERENT 2D TRANSFORM HARDWARE DESIGNS BASED ON FPGA

Design	**[14]**	**[16]**	**[15]**	**[17]**	**[8]**
Standard	JEM	JEM	JEM	HEVC	HEVC
Technology/Platform	FPGA 40nm	Mid-End FPGA 20nm	Mid-End FPGA 20nm	Xilinx Zynq	Stratix V GX
Maximum Frequency (MHz)	167	458	147	222	120
Pixel Parallelism (pixels/cycle)	8	16/N	N/4	4	8–16
ALMs	7,930	999	133,017	5,806	17,755
DSPs	-	32	1561	128	354
Transform Unit (M×N)	M = N = 4, 8.	M = N = 4, 8, 16, 32.	M = 4, 8, 16, 32; N = 4, 8, 16, 32.	M = N = 4, 8, 16, 32.	M = N = 4, 8, 16, 32.
Transform Type	DCT-II, DCT-V, DCT-VIII, DST-I, DST-VII			DCT-II	DCT-II

TABLE V
COMPARISONS OF TRANSPOSE MEMORY DESIGNS

Memory Design	**[3]**	**[7]**	**Proposed**	
			SRAM	**Register Array**
Number of SRAMs/ Register Arrays	4	4	32	
Width (bits)	512	16	16	
Depth	8	256	64	
Single area (μm^2)	68,431	20,247	6,677.7	12,583
Total area (μm^2)	273,724	80,988	213,664	402,656

between these two areas). This makes our area much larger than that of [7].

V. CONCLUSION

This paper proposes a high-performance pipelined 2D transform architecture for DST-VII/DCT-VIII. For a perfect pipeline, a 32×32 range-based pipeline scheme is employed, and 32 transformed coefficients can be calculated per clock cycle after a certain delay. To reduce hardware cost, the RAG-n algorithm is adopted to design the multiplier blocks. The experimental results illustrate that the SAU-based design can save the area and power consumption by up to 73.1% and 9.9%, respectively, compared with other works. When SRAM is used, the memory area can be reduced by at least 21.9%.

This paper is aimed at DST-VII and DCT-VIII operations, and DCT-II is currently not considered. The proposed architecture can be easily extended to a unified one only with an additional DCT-II engine. To further reduce the hardware cost, we will consider a reconfigurable architecture and the approximation scheme in the future so that the three transform types can share an architecture under the premise of ensuring the coding performance as much as possible.

REFERENCES

[1] *Working Draft 4 of Versatile Video Coding*, document JVET M1001-v7, Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC (JTC 1/SC 29/WG 11), Jan. 2019.

[2] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[3] 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, Jul. 2012, pp. 788–793.

[4] 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.

[5] W.-H. Chen, C. Smith, and S. Fralick, "A fast computational algorithm for the discrete cosine transform," *IEEE Trans. Commun.*, vol. 25, no. 9, pp. 1004–1009, Sep. 1977.

[6] T. Ma, C. Liu, Y. Fan, and X. Zeng, "A fast 8×8 IDCT algorithm for HEVC," in *Proc. IEEE 10th Int. Conf. ASIC*, Shenzhen, China, Oct. 2013, pp. 1–4.

[7] H. Sun, D. Zhou, J. Zhu, S. Kimura, and S. Goto, "An area-efficient 4/8/16/32-point inverse DCT architecture for UHDTV HEVC decoder," in *Proc. IEEE Vis. Commun. Image Process. Conf.*, Valletta, Malta, Dec. 2014, pp. 197–200.

[8] Y. Zhang and C. Lu, "Efficient algorithm adaptations and fully-parallel hardware architecture of H.265/HEVC intra encoder," *IEEE Trans. Circuits Syst. Video Technol.*, to be published.

[9] E. Kalali, A. C. Mert, and I. Hamzaoglu, "A computation and energy reduction technique for HEVC discrete cosine transform," *IEEE Trans. Consum. Electron.*, vol. 62, no. 2, pp. 166–174, May 2016.

[10] H. Sun, Z. Cheng, A. M. Gharehbaghi, S. Kimura, and M. Fujita, "Approximate DCT design for video encoding based on novel truncation scheme," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 4, pp. 1517–1530, Apr. 2019.

[11] G. Pastuszak and A. Abramowski, "Algorithm and architecture design of the H.265/HEVC intra encoder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 26, no. 1, pp. 210–222, Jan. 2016.

[12] A. Kammoun, S. Ben Jdidia, F. Belghith, W. Hamidouche, J. F. Nezan, and N. Masmoudi, "An optimized hardware implementation of 4-point adaptive multiple transform design for post-HEVC," in *Proc. 4th Int. Conf. Adv. Technol. Signal Image Process.*, Sousse, Tunisia, Mar. 2018, pp. 1–6.

[13] S. Ben Jdidia, A. Kammoun, F. Belghith, and N. Masmoudi, "Hardware implementation of 1-D 8-point adaptive multiple transform in post-HEVC standard," in *Proc. 18th Int. Conf. Sci. Techn. Autom. Control Comput. Eng.*, Monastir, Tunisia, Dec. 2017, pp. 146–151.

[14] A. C. Mert, E. Kalali, and I. Hamzaoglu, "High performance 2D transform hardware for future video coding," *IEEE Trans. Consum. Electron.*, vol. 62, no. 2, pp. 117–125, May 2017.

[15] A. Kammoun, W. Hamidouche, F. Belghith, J.-F. Nezan, and N. Masmoudi, "Hardware design and implementation of adaptive multiple transforms for the versatile video coding standard," *IEEE Trans. Consum. Electron.*, vol. 64, no. 4, pp. 424–432, Nov. 2018.

[16] M. J. Garrido, F. Pescador, M. Chavarrías, P. J. Lobo, and C. Sanz, "A high performance FPGA-based architecture for the future video coding adaptive multiple core transform," *IEEE Trans. Consum. Electron.*, vol. 64, no. 1, pp. 53–60, Feb. 2018.

[17] M. Chen, Y. Zhang, and C. Lu, "Efficient architecture of variable size HEVC 2D-DCT for FPGA platforms," *AEU-Int. J. Electron. Commun.*, vol. 73, pp. 1–8, Mar. 2017.

[18] P. Sjövall, V. Viitamäki, J. Vanne, and T. D. Hämäläinen, "High-level synthesis implementation of HEVC 2-D DCT/DST on FPGA," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, New Orleans, LA, USA, Mar. 2017, pp. 1547–1551.

[19] Q. Shang, Y. Fan, W. Shen, S. Shen, and X. Zeng, "Single-port SRAM-based transpose memory with diagonal data mapping for large size 2-D DCT/IDCT," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 11, pp. 2423–2427, Nov. 2014.

[20] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Trans. Algorithms*, vol. 3, no. 2, p. 11, May 2007.

[21] *VTM Reference Software*. Accessed: Mar. 5, 2019. [Online]. Available: https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/release