Architecture for Area-Efficient 2-D Transform in H.264/AVC

Yu-Ting Kuo, *Tay-Jyi Lin*, *Chih-Wei Liu*, and *Chein-Wei Jen*

Department of Electronics Engineering
National Chiao Tung University, Taiwan

Abstract

As the VLSI technology advances continuously, ASIC can easily achieve the required performance and most of them are actually over-designed. Thus, architecture shrinking is inevitable in optimal designs especially when supply voltages are getting lower. However, conventional designs starting from minimization of algorithmic operations (e.g. multiply) may not always lead to optimal architectures, for the wires and the interconnection complexity significantly grow and have become predominant. This paper explores algorithms and architectures for the 2-D transform in H.264/AVC, of which the operations are very simple (i.e. only shift and add). We have shown that fewer operations do not always result in more compact designs. In our experiments with the UMC 0.18μm CMOS technology, the most straightforward matrix multiplication without separable 2-D operation or any fast algorithm has the best area efficiency for D1-size (720×480) video at 30fps. It saves 48%, 34%, and 16% silicon area of the previous works respectively.

1. Introduction

The H.264/AVC [1] is a new video coding standard, which is developed for efficient video compression and reliable data transport. It provides better performance over its prior standards such as H.263 and MPEG-4, and the applications include videoconferencing, video telephony, digital TV and DVD, etc. There are four kinds of 4-by-4 2-D transforms in the H.264/AVC video encoder – forward, inverse, Hadamard, and inverse Hadamard transforms. These four transforms are very similar and therefore this paper discusses the forward transform only, which is approximation of the 2-D discrete cosine transform (DCT) [2] with the scaling multiplications integrated into the quantizer.

There have been quite a lot of works on the algorithms and the architectures for DCT such as [3][4], but only few focus on DCT with small block sizes, such as those for H.264/AVC [5][6][7]. Besides, it is not the functional units but the interconnections that dominate the circuit performance (i.e. speed, silicon area, and even power consumption) in today's VLSI technology. That is, choosing an algorithm with fewer operations may not always lead to less hardware complexity as before. Moreover, the 2-D transforms in H.264/AVC can be carried out with only additions and shifts, and such simple operations make the interconnection overheads much more significant. This paper discusses the area efficiency of the architecture mapping of several algorithms for the forward transform, and we have shown that fewer operations do not necessarily result in smaller designs. In our experiments with the 0.18μm CMOS technology, the most straightforward matrix multiplication without separable 2-D operation or fast algorithm has the best area efficiency for D1-size (720×480) video at 30fps. It can save 48%, 34%, and 16% silicon area of the previous works [5][6][7] respectively.

The rest of this paper is organized as follows. Section 2 first reviews the algorithms for the 2-D transform in H.264/AVC. Section 3 then describes their architecture mapping. Several designs are implemented via effective architecture mapping of the algorithms, and the results and area comparison are available in Section 4. Section 5 concludes this paper.

This research was supported in part by the National Science Council under Grant NSC93-2220-E-009-017 and the Ministry of Economic Affairs under Grant 94-EC-17-A-01-S1-034.

2. Algorithms for Forward Transform

2.1 Separable or direct 2-D transforms

The forward transform in H.264/AVC is defined as

$$Y = CXC^T = MC^T \tag{1}$$

where X and Y denote the 4-by-4 input and output matrices respectively. M denotes the intermediate result matrix and the coefficient matrix C is defined as

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}.$$

$M=CX$ represents the four column-wise 1-D transforms as

$$\begin{bmatrix} m_{0,j} \\ m_{1,j} \\ m_{2,j} \\ m_{3,j} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} x_{0,j} \\ x_{1,j} \\ x_{2,j} \\ x_{3,j} \end{bmatrix} \tag{2}$$

on the input matrix X, while $Y=MC^T$ represents another four similar row-wise 1-D transforms on the intermediate matrix M. Therefore, Eq. (1) is called the "separable" 2-D transform.

Alternatively, we can rewrite Eq. (1) as Eq. (3), and compute the output matrix Y directly by performing the matrix-vector multiplication on the elements of X. This is the direct 2-D transform.

$$\begin{bmatrix} y_{00} \\ y_{01} \\ y_{02} \\ y_{03} \\ y_{10} \\ y_{11} \\ y_{12} \\ y_{13} \\ y_{20} \\ y_{21} \\ y_{22} \\ y_{23} \\ y_{30} \\ y_{31} \\ y_{32} \\ y_{33} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 & 2 & 1 & -1 & -2 & 2 & 1 & -1 & -2 & 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 & 1 & -2 & 2 & -1 & 1 & -2 & 2 & -1 & 1 & -2 & 2 & -1 \\ 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -2 & -2 & -2 & -2 \\ 4 & 2 & -2 & -4 & 2 & 1 & -1 & -2 & -1 & -1 & 1 & 2 & -4 & -2 & 2 & 4 \\ 2 & -2 & -2 & 2 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -2 & 2 & 2 & -2 \\ 2 & -4 & 4 & -2 & 1 & -2 & 2 & -1 & -1 & 2 & -2 & 1 & -2 & 4 & -4 & 2 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 & -2 & -1 & 1 & 2 & -2 & -1 & 1 & 2 & 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 & -1 & 2 & -2 & 1 & -1 & 2 & -2 & 1 & 1 & -2 & 2 & -1 \\ 1 & 1 & 1 & 1 & -2 & -2 & -2 & -2 & 2 & 2 & 2 & 2 & -1 & -1 & -1 & -1 \\ 2 & 1 & -1 & -2 & -4 & -2 & 2 & 4 & 4 & 2 & -2 & -4 & -2 & -1 & 1 & 2 \\ 1 & -1 & -1 & 1 & -2 & 2 & 2 & -2 & 2 & -2 & -2 & 2 & -1 & 1 & 1 & -1 \\ 1 & -2 & 2 & -1 & -2 & 4 & -4 & 2 & 2 & -4 & 4 & -2 & -1 & 2 & -2 & 1 \end{bmatrix} \begin{bmatrix} x_{00} \\ x_{01} \\ x_{02} \\ x_{03} \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{20} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{30} \\ x_{31} \\ x_{32} \\ x_{33} \end{bmatrix} \tag{3}$$

Since the coefficients are very simple and only require trivial multiplications by shifts, the computational complexities of these two algorithms can thus be compared by the number of the required additions and subtractions. First, the separable 2-D forward transform needs eight 1-D transforms shown in Eq. (2), each of which requires twelve additions/subtractions. Therefore, it needs 96 additions/subtractions in total. On the other hand, the direct 2-D forward transform requires 240 additions/ subtractions.

2.2 Algorithmic strength reduction

Both 1-D and direct 2-D forward transform algorithms have symmetric coefficients in their transform matrices, which can be exploited to rearrange the computations and to effectively reduce the additions/subtractions. For example, the 1-D forward transform can save four additions as shown in Fig. 1. Thus, the separable 2-D and fast forward transform, which is adopted in [5] and [6], requires only 64 additions/subtractions.

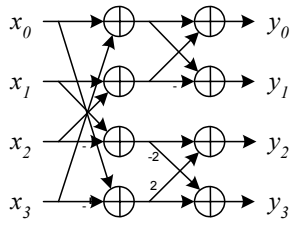

Fig 1. Fast 1-D transform

Similarly, the symmetry property of the direct 2-D forward transform can also be exploited by rewriting Eq. (3) as

$$\begin{bmatrix} \bar{y}_0 \\ \bar{y}_2 \end{bmatrix} = \begin{bmatrix} C & C \\ C & -C \end{bmatrix} \begin{bmatrix} \bar{x}_0 + \bar{x}_3 \\ \bar{x}_1 + \bar{x}_2 \end{bmatrix} \quad \begin{bmatrix} \bar{y}_1 \\ \bar{y}_3 \end{bmatrix} = \begin{bmatrix} 2C & C \\ C & -2C \end{bmatrix} \begin{bmatrix} \bar{x}_0 - \bar{x}_3 \\ \bar{x}_1 - \bar{x}_2 \end{bmatrix} \quad (4)$$

where $\bar{x}_i$ and $\bar{y}_i$ are the transposes of the *i*-th rows of the input and the output matrices respectively [7]. Interestingly, this direct 2-D and fast forward transform also requires 64 additions/subtractions after the computation rearrangements. Table 1 summarizes the computational complexities of the above four forward transform algorithms.

Table 1 Comparison of forward transform algorithms

Algorithm		# Additions	
2-D Transform	Strength Reduction		
Separable	Direct	96	
	Fast	64	[5][6]
Direct	Direct	240	
	Fast	64	[7]

3. Architecture Mapping

The most straightforward method to translate an algorithm into its hardware implementation is to allocate a dedicated functional unit to each operation. However, it is too costly for many practical applications, especially when advanced fabrication technology is used. Architecture shrinking that time-multiplexes several operations on shared resources is a commonly used technique to reduce such unnecessary waste. Folding [8] is the systematic methodology that maps DSP algorithms on hardware architectures. To clarify our further discussions, we classify the architectures into two major categories – ***data-parallel*** and ***data-serial***. The former handles multiple input data concurrently, while the latter processes a single input datum at one time. Besides, we will extensively use the term ***scaling-down factor*** (SF) to indicate the number of steps for the target architecture to perform the 4-by-4 forward transform. For example, the aforementioned "most straightforward architecture mapping" that dedicates a functional unit to each operation has SF=1. Note that a higher SF may only imply that fewer functional units are required in the hardware implementation. Indeed, it incurs extra multiplexers and sometimes additional registers. These overheads will compensate the benefits, not to mention the fact that interconnection predominates the performance in today's deep-submicron VLSI technology.

3.1 Data-parallel architectures

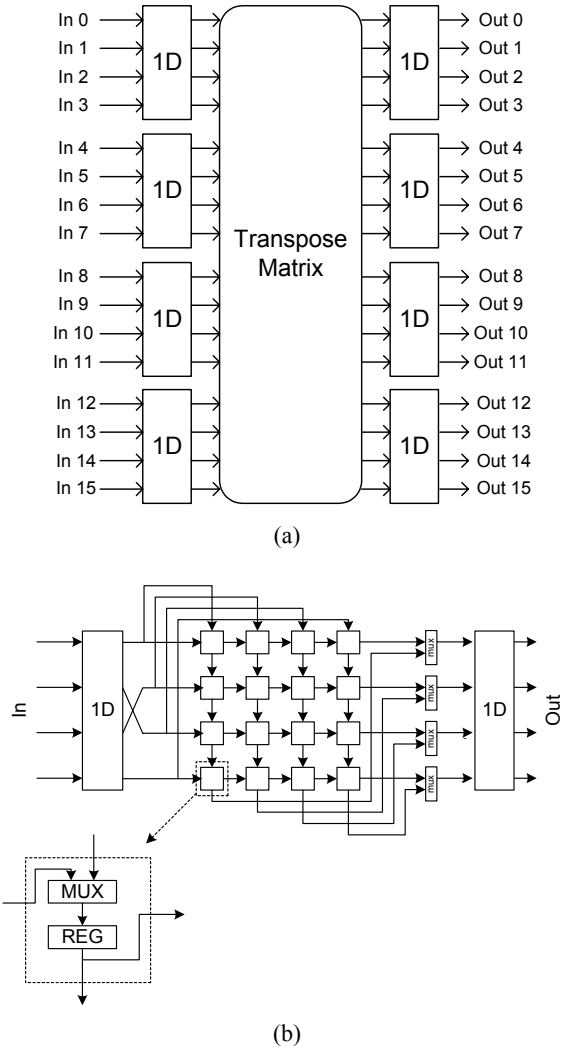

Fig 2. Data-parallel architectures for separable 2-D/fast forward transform (a) SF=1 [5], (b) SF=4 [6]

Fig. 2(a) depicts the most straightforward hardware mapping of the separable 2-D and fast forward transform algorithm [5]. It contains eight 1-D fast transform modules shown in Fig. 1. A direct interconnection network is included for transposition of the intermediate results. Fig. 2(b) shows a shrunk version of Fig. 2(b) with SF=4. Therefore, it requires only two 1-D transform modules, and the four column- and the four row-transforms are performed respectively in the corresponding modules. Besides, the original memory-less transpose matrix becomes a transpose memory with 16 registers after the architecture shrinking.

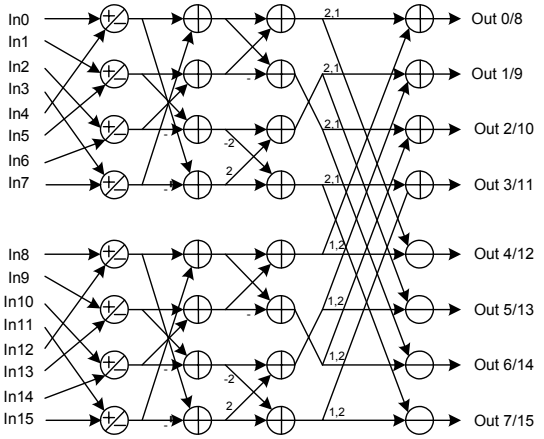

Fig 3. Data-parallel architecture for direct 2-D/fast forward transform algorithm [8]

Fig. 3 shows the architecture mapping of the direct 2-D and fast forward transform algorithm in Eq. (4) with SF=2 [7]. For each 4-by-4 forward transform, the 16 input samples hold for two cycles, each of which generates half output results. The architecture shrinking in this case is very efficient, which just incurs an additional output de-multiplexer.

3.2 Data-serial architectures

Data-serial architectures process one input sample at a time and thus they would have SF>16 for the 4-by-4 2-D forward transform. Fig. 4(a) shows the data-serial architecture for the direct 2-D and direct forward transform algorithm in Eq. (3) with SF=16. This architecture processes an input sample in a cycle, which multiplies the sample with 16 coefficients in a column of the 16-by-16 coefficient matrix in Eq. (3) and accumulates the 16 products into the 16 registers respectively. Thus, the accumulation registers will have the 16 outputs of the forward transform after 16 cycles.

Fig. 4(b) shows our proposed area-efficient architecture for the 2-D forward transform in H.264/AVC with SF=256. This architecture is actually a shrunk design from Fig. 4(a) by folding it 16 times. Here, each sample holds at the input for 16 cycles, and is multiplied by 16 coefficients accordingly and then accumulated in the 16 output registers. After 256 cycles, the outputs will be ready on these 16 accumulation registers.

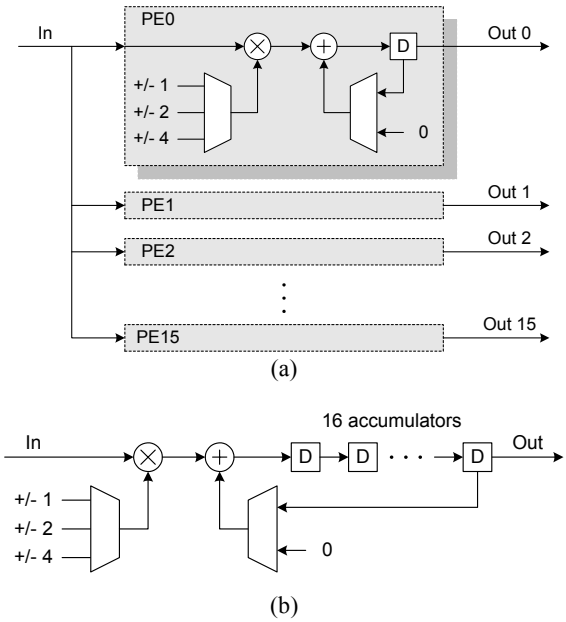

Fig 4. Data-serial architectures for direct 2-D/direct forward transform algorithm (a) SF=16, (b) SF=256

4. Simulation Results

In this section, we will evaluate the area efficiency of our proposed data-serial architecture for the 4-by-4 2-D forward transform in H.264/AVC with some typical designs based on the algorithms and architectures classified in Section 3 and 4. The designs under investigation are listed in Table 2 with the abbreviated names. First, D/D/S/256 represents the proposed area-efficient design depicted in Fig. 4(b), while S/F/P/1, S/F/P/4 and D/F/P/2 denote the three previous works [5], [6] and [7] respectively. Data-serial architectures for the separable and the direct 2-D fast algorithms are derived for reference by applying ASAP operation scheduling and the forward-backward register allocation with minimum registers [8]. These two designs are S/F/S/64 and D/F/S/64, and both of them only have a single adder as our proposed D/D/S/256. An additional D/F/P/1 is designed to evaluate the architecture shrinking of D/F/P/2 [7], and to provide a fair comparison with S/F/P/1 [5]. Finally, D/D/S/128 is constructed to show the performance scalability of our proposed architecture.

Table 2 List of compared architectures

Algorithm		Architecture	SF	Notation
2-D Transform	Strength Reduction			
Separable	Fast	Parallel	1	S/F/P/1 [5]
		Parallel	4	S/F/P/4 [6]
		Serial	64	S/F/S/64
Direct		Parallel	1	D/F/P/1
		Parallel	2	D/F/P/2 [7]
		Serial	64	D/F/S/64
	Direct	Serial	128	D/D/S/128
		Serial	256	**D/D/S/256**

The eight designs in Table 2 are first described in Verilog RTL and synthesized using Synopsis Design Compiler. The cell library is from Artisan and it is designed for the UMC 0.18µm CMOS technology. The clock period constraint is derived from the required pixel processing rate. For example, a D1-size image has 720×480×1.5=518,400 pixels in 4:2:0 color format. The pixel count amounts to 15,552,000 for 30 frames. D/D/S/256 processes 16 samples in 256 clock cycles, and its allowable cycle period for D1@30fps is

$$\frac{1\times 16}{15{,}552{,}000\times 256} = 4.019 \text{ ns.}$$

The clock period constraints and the minimum area reported by the Synopsys Design Compiler for the eight designs under investigation are shown in Table 3. The designs are listed in the descending order of their reported area.

Table 3 Synthesis results for D1@30fps

Architecture	Clock Period	Area
S/F/P/1 [5]	1,024 ns	199,323.1
D/F/P/1	1,024 ns	198,464.3
S/F/S/64	16 ns	179,337.8
D/F/S/64	16 ns	159,040.1
S/F/P/4 [6]	256 ns	158,051.3
D/F/P/2 [7]	512 ns	123,927.9
D/D/S/128	8 ns	107,794.5
D/D/S/256	**4 ns**	**104,495.0**

Thanks to the fast functional unit supported by the advanced 0.18µm technology and the regular data-serial architecture, D/D/S/256 and D/D/S/128 are the two designs with the smallest area, despite the required operations are 3.75 times of those with fast algorithms. Both S/F/S/64 and D/F/S/64 have the same single adder, but their synthesis results are even worsen than S/F/P/4 [6] and D/F/P/2 [7] with much more functional units. In other words, the fast algorithms reduce the operations at the cost of irregular dataflow, which prevent their architectures from efficiently scaling down.

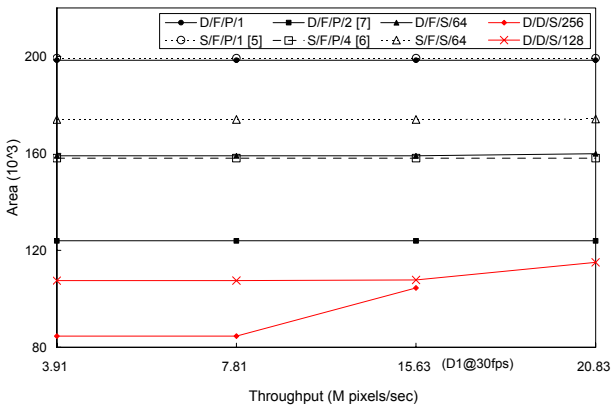

Fig. 5. Comparison of DCT algorithms and architectures

Fig. 5 shows the minimum area for the eight designs under different throughput requirements. The report numbers are almost constants due to the loose synthesis constraints derived from the pixel rates of interests, except D/D/S/256 and D/D/S/128. The processing capability of D/D/S/256 is up to D1@30fps in the UMC 0.18µm technology, and it stands for the most area-efficient design for the applications with lower pixel rates. By the way, the separable 2-D transform seems unable to perform well for small block sizes, for D/F/P/1 and D/F/P/2 always outperform S/F/P/1 and S/F/P/4 respectively.

5. Conclusions

This paper reviews the algorithms and architectures for the 4-by-4 2-D forward transform in H.264/AVC and it describes an area-efficient data-serial architecture for the transform. Owing to the fast functional unit in the advanced 0.18µm technology and the regular dataflow, the proposed design of direct matrix multiplication without any fast algorithm or separable 2-D operations stands for the most area-efficient one for applications with lower pixel rates than D1@30fps. It can save 48%, 34%, and 16% silicon area of the previous works [5][6][7] respectively. Our experimental results in the UMC 0.18µm CMOS technology show that separable 2-D transform algorithms seem unable to perform well for small block sizes. Therefore, high-performance applications such as HDTV and cinema videos may adopt the direct 2-D and fast forward transform algorithm [7]. For low-cost and area-critical codec, the proposed data-serial architecture with the direct 2-D and direct forward transform algorithm is an effective alternative. By the way, although our proposed design is the most area-efficient one for many practical applications (throughput < D1@30fps), the required clock rate (250MHz) is extremely high. It wastes significant power and may not be acceptable in many embedded systems. We are studying the power issues and trying to identify the application ranges that our proposed approach has low-power advantages. In the future, we will study circuit techniques to reduce the clock overheads to broaden its applications.

References

[1] I. E. G. Richardson, *H.264 and MPEG-4 Video Compression*, Wiley, 2003

[2] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantage, Applications*, Academic Press, 1990

[3] S. F. Hsiao and Y. H. Hu, "High-radix low-complexity architectures for long-length DCT using conventional arithmetic and ROM-based distributed arithmetic," in *Proc. VLSI-TSA*, 2003

[4] M. Puchel, "Cooley-Tukey FFT like algorithms for the DCT," in *Proc. ICASSP*, 2003

[5] K. H. Chen, J. I. Guo, K. C. Chao, J. S. Wang and Y. S. Chu, "A high performance low power direct 2-D transform coding IP design for MPEG-4 AVC/H.264 with a switching power suppression technique," in *Proc. VLSI-TSA-DAT*, 2005

[6] T. C. Wang, Y. W. Huang, H. C. Fang, and L. G. Chen, "Parallel 4x4 2D transform and inverse transform architecture for MPEG-4 AVC/H.264," in *Proc. ISCAS*, 2003

[7] Z. Y. Cheng, C. H. Chen, B. D. Liu, and J. F. Yang, "High throughput 2-D transform architectures for H.264 advanced video coders," in *Proc. APCCAS*, 2004

[8] K. K. Parhi, *VLSI Digital Signal Processing Systems Design and Implementation,* Wiley, 1999