AN EFFICIENT FREQUENCY DOMAIN INTRA PREDICTION FOR H.264/AVC

Mohsen Shaaban and Magdy Bayoumi

The Center for Advanced Computer Studies, University of Louisiana at Lafayette
{mmm5554, mab}@cacs.louisiana.edu

ABSTRACT

This paper presents an efficient intra prediction implementation for H.264/AVC in the frequency domain. Intra prediction in the frequency domain is vital for transform domain heterogeneous video transcoding in wireless and mobile networks. A limited computation capabilities constraint constitutes a burden over such networks. New distributed intra prediction arithmetic is proposed with only addition and shift operations for transform domain intra prediction modes computations. Compared to previous attempts the proposed method reduces the computations extensively by eliminating the expensive matrix multiplications and omits the need for excessive memory storage.

Index Terms— Intra Prediction, Transform Domain, Distributed Arithmetic, H.264/AVC, Video Transcoding

1. INTRODUCTION

The great advances in mobile multimedia devices and wireless video communication have laid the ground for a wide spread of new applications and systems. Recently, this bursts the emergence of multimedia transcoding in mobile and wireless networks. Video Transcoding [1] is focused on filling up the gap between the high resource requirements of video data and the limited bandwidth and computational capabilities offered by wireless networks and handheld devices. Bit-rate reduction, spatial/temporal resolution reduction and format/standard conversion are the main applications behind video transcoding.

H.264/AVC [2] were jointly designed by ITU and ISO for low to high bit rate applications with very high coding efficiency at 1/3-1/2 the bit rate of the previous standards. Heterogeneous Video Transcoding from other standards to H.264 has gained more importance in the recent years [3, 4]. Moreover, frequency domain video transcoding was shown to be computationally efficient than pixel domain transcoding [5] as it eliminate the need for inverse transform/transform operations. H.264 defines intra prediction in the pixel domain. Transform domain intra prediction was investigated in [6] but it involved a considerable amount of matrix multiplications and additional memory storage.

We adopt a new distributed arithmetic method (NEDA) described by the authors in [7] to reduce the computation complexity extensively by eliminating the expensive matrix multiplication completely. Only addition and shift operations are required for transform domain intra prediction modes computations. Moreover it omits the need for excessive memory storage appointing it for frequency domain video transcoding in wireless and mobile networks.

This paper is organized as follows. In section 2, a brief overview of NEDA is provided. The proposed transform domain intra prediction is described in section 3. Computation complexity and conclusion are presented in sections 4 and 5 respectively.

2. NEW DISTRIBUTED ARITHMETIC (NEDA)

Distributed Arithmetic (DA) has achieved a large scale application in Digital Signal Processing (DSP) architectures [8]. The main advantage of DA approach is that it speeds up the multiplication process by pre-computing all possible product values and storing these values in memory. The input data can then be exploited to directly address the memory and the result. A drawback is that memory size grows exponentially with number of inputs and internal precision.

A NEw Distributed Arithmetic (NEDA) was proposed in a previous work to implement the inner product of vectors in the form of 2's complement numbers using only additions followed by a small number of shifts at the final stage. The architecture using NEDA is free of memory, multipliers and subtraction, making it very suitable for computation/area sensitive systems.

In the following we give a brief derivation to NEDA, due to page limit interested reader is referred to [7] for full details. Consider the following matrix product:

$$Y = [A_1 A_2 \cdots A_k \cdots A_L] \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_k \\ \vdots \\ X_L \end{bmatrix} \quad (1)$$

A_k is in 2's complement format and can be expressed as:

$$A_k = -A_k^M 2^M + \sum_{i=N}^{M-1} A_k^i 2^i \quad (2)$$

Where $A_k^i = 0 \; or \; 1;\; i = N, N+1, \cdots, M$ and A_k^M is the sign bit, and A_k^N is the least significant bit. X_k is the input data words. Unlike DA, we distribute the bits of the fixed coefficients in NEDA not the bits of the frequently varying inputs. *(M-N+1)* is the DA *precision* of NEDA.

Combining (1) and (2), we get:

$$Y = [2^N 2^{N+1} \cdots 2^k \cdots 2^M] \left\{ \begin{bmatrix} A_1^N & A_2^N \cdots & A_k^N & \cdots & A_L^N \\ & \vdots & & \ddots & \vdots \\ -A_1^M & -A_2^M \cdots & -A_k^M & \cdots & -A_L^M \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_k \\ \vdots \\ X_L \end{bmatrix} \right\} \quad (3)$$

Matrix $[A]$ in (3) is crucial to NEDA since its structure can lead to savings in computations. It only consists of 0's and 1's. Computation of Y in (3) involves only addition operations. The final stage in (3) can be realized by small number of shift operations. We donate $[A]$ as the adder array.

3. TRANSFORM DOMAIN INTRA PREDICTION

In addition to inter prediction used in all video coding standards, H.264 achieves more compression efficiency by using neighboring samples of previously coded blocks to predict current blocks in a process known as intra prediction. Two common intra prediction modes are used in H.264, Intra_4x4 and Intra_16x16. Intra_4x4 support 9 prediction directions defined based on the spatial activity between blocks. Intra_16x16 support 4 prediction directions. Intra prediction modes and directions are selected based on minimizing block rate-distortion value.

In the following sub-sections, we incorporate NEDA into the derivation of transform domain intra prediction.

3.1. Intra_4x4

Each 4x4 block is predicted from spatially neighboring samples as shown in Fig. 1a. Block pixel's a-p is predicted from border coded samples in adjacent blocks (A-Q). The nine prediction directions for each block are shown in Fig. 1b. We draw the analysis for selected prediction directions; other directions follow the same analogy.

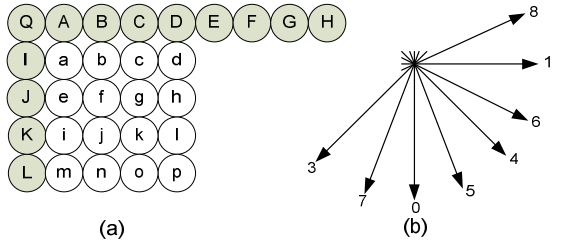

Figure 1. (a) Intra_4x4 prediction. (b) Prediction directions

3.1.1. Mode 0: Vertical Prediction

In this prediction mode pixels of each of the four columns of the current block are predicted from the neighboring pixels A-D respectively. The predicted block can be expressed in matrix form as:

$$\begin{bmatrix} A & B & C & D \\ A & B & C & D \\ A & B & C & D \\ A & B & C & D \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ A & B & C & D \end{bmatrix} \quad (4)$$

where 'x' donate "don't care" pixels in the neighboring block. Applying H.264 transform to both sides of (4):

$$HT\left(\begin{bmatrix} A & B & C & D \\ A & B & C & D \\ A & B & C & D \\ A & B & C & D \end{bmatrix}\right) = \begin{bmatrix} 1 & -1.265 & 1 & -0.632 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} HT\left(\begin{bmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ A & B & C & D \end{bmatrix}\right) \quad (5)$$

We donate the second matrix as the fixed coefficients matrix [C]. Equation (5) implies that intra prediction can be achieved in the transform domain but it requires 8 multiplications and 12 additions for such simple mode. Besides, [C] has to be pre-computed and stored in memory. Number of multiplications and memory storage grows much higher for more complex modes as we'll observe later.

We begin our justification for using NEDA with discussing the generation of the adder arrays by distributing the bits of the coefficients in [C]. First, each transform coefficient of the predicted block's first column is calculated as: (other columns will follow the same analysis)

$$P_u = \sum_{k=0}^{3} [A_u(k)] \,.\, p(k) \quad (6)$$

Where $A_u = [A_u(k)]$ represent the first row vector in the fixed coefficient matrix, $k = 0 \rightarrow 3$ and $u = 0 \rightarrow 3$; and $p(k)$ represent the k^{th} coefficient of the coded input neighboring block's first column vector.

Note that equation (3) is specified for either integer or fraction coefficients but not both at the same time, whereas, equation (5) has [C] with both integers and fractions coefficients. Therefore, we modify NEDA by expanding (3) to accommodate both integer and fractional coefficients as:

$$Y = Y_{int} + Y_{fr} \quad (7)$$

$$Or,$$

$$Y = \left[2^N 2^{N+1} \cdots 2^M 2^{-M} \cdots 2^{-(N+1)} 2^{-N}\right] . \left\{ \begin{bmatrix} A_{i1}^N & A_{i2}^N \cdots & A_{ik}^N & \cdots & A_{iL}^N \\ & \vdots & & \ddots & \vdots \\ -A_{i1}^M & -A_{i2}^M \cdots & -A_{ik}^M & \cdots & -A_{iL}^M \\ A_{f1}^N & A_{f2}^N \cdots & A_{fk}^N & \cdots & A_{fL}^N \\ & \vdots & & \ddots & \vdots \\ -A_{f1}^M & -A_{f2}^M \cdots & -A_{fk}^M & \cdots & -A_{fL}^M \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_k \\ \vdots \\ X_L \end{bmatrix} \right\} \quad (8)$$

Where A_i and A_f are the integer and fractional adder arrays.

And, we rewrite [C] as the sum of its integer and fractional parts as:

$$[C] = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -0.265 & 0 & -0.632 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad (9)$$

Now from (6), and if DA precision is chosen to be 11 bits, then we have P_{fu} (for the fractional part) as in (10):

$$P_{fu} = \begin{bmatrix} P_{fu}(-10) \\ P_{fu}(-9) \\ \vdots \\ \vdots \\ P_{fu}(2) \\ P_{fu}(1) \\ P_{fu}(0) \end{bmatrix} = A_{fu} \,.\, \begin{bmatrix} p(0) \\ p(1) \\ p(2) \\ p(3) \end{bmatrix} \quad (10)$$

$$A_{i0} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad A_{f0} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad (11)$$

Note that $P_{fu}(0)$ is the sign bit and $P_{fu}(-10)$ is the LSB.

Expressed in its 11 bit DA form, we have $A_{f0} = \left[A_{f0}(0)\ A_{f0}(1)\ A_{f0}(2)\ A_{f0}(3)\right]$ as the adder array of P_{f0} as in (11). $A_{f0}(0) \rightarrow A_{f0}(3)$ represent the 2's complement of the coefficients in the first row of the fractional part in [C]. The bottom row of A_{f0} is the sign bits of $A_{f0}(k)$ and the top row is the LSBs.

Note that matrix [A] is a sparse matrix in most cases due to the inherent nature of DA representation of numbers. A direct implementation of A_{f0} on a row-by-row basis will require 4 additions whereas by using our compression scheme in [7], which ensures that common intermediate operations can be shared; only 1

addition is required. The butterfly structure after compression is shown in Fig. 2b.

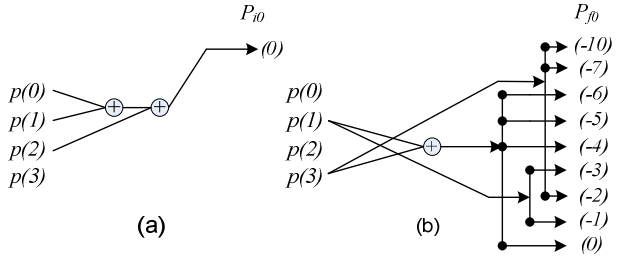

Figure 2. Butterfly structures for Mode 0 (a) Integer (b) Fraction

For P_{iu} (the integer part), one can show that it only require 2 addition as shown in Fig. 2a since A_{i0} as in (11) have 1's only in its first row.

From the above analysis, only 3 additions are needed to produce the first transform coefficient. A total of 12 additions and 10 shifts are required to compute this mode's prediction sample of the current block in the transform domain with no need for any multiplication operation or memory storage.

3.1.2. Mode 1: Horizontal Prediction

The prediction block following this mode is predicted from the neighboring pixels I-L and has the form of:

$$HT\left(\begin{bmatrix} I & I & I & I \\ J & J & J & J \\ K & K & K & K \\ L & L & L & L \end{bmatrix}\right) = HT\left(\begin{bmatrix} x & x & x & I \\ x & x & x & J \\ x & x & x & K \\ x & x & x & L \end{bmatrix}\right)\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1.265 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ -0.632 & 0 & 0 & 0 \end{bmatrix}, \quad (12)$$

which has the same complexity analysis as (5).

3.1.3. Mode 2: DC Prediction

The predicted pixels of this mode are the average of the border pixels of the above and to the left blocks:

$$\frac{1}{8}(A+B+C+D)\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} + \frac{1}{8}(I+J+K+L)\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}. \quad (13)$$

The H.264 integer transform of (12) is expressed as:

$$\frac{1}{8}HT\left(\begin{bmatrix} A & B & C & D \\ A & B & C & D \\ A & B & C & D \\ A & B & C & D \end{bmatrix}\right)\begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \frac{1}{8}HT\left(\begin{bmatrix} I & I & I & I \\ J & J & J & J \\ K & K & K & K \\ L & L & L & L \end{bmatrix}\right)\begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (14)$$

From (14), we can see that the DC prediction required computations that already been done in the Vertical and Horizontal modes. Therefore, only one addition is required to compute the prediction samples of the current block in the transform domain.

3.1.4. Mode 4: Diagonal_Down/Right Prediction

This mode is the most complicated among all modes. According to [6], it requires 232 multiplications and 200 additions per block and a considerable number of pre-computed matrices to be stored in memory. We analyze this mode to show the amount of saving in computations using NEDA.

Using matrix manipulation, the predicted samples values can be split according to predictor pixels from the neighboring (above, corner and left) blocks:

$$\frac{1}{4}\begin{bmatrix} A+2Q+I & Q+2A+B & A+2B+C & B+2C+D \\ Q+2I+J & A+2Q+I & Q+2A+B & A+2B+C \\ I+2J+K & Q+2I+J & A+2Q+I & Q+2A+B \\ J+2K+L & I+2J+K & Q+2I+J & A+2Q+I \end{bmatrix} =$$

$$\frac{1}{4}\left(\begin{bmatrix} A & 2A+B & A+2B+C & B+2C+D \\ 0 & A & 2A+B & A+2B+C \\ 0 & 0 & A & 2A+B \\ 0 & 0 & 0 & A \end{bmatrix} + \begin{bmatrix} 2Q & Q & 0 & 0 \\ Q & 2Q & Q & 0 \\ 0 & Q & 2Q & Q \\ 0 & 0 & Q & 2Q \end{bmatrix} + \begin{bmatrix} I & 0 & 0 & 0 \\ 2I+J & I & 0 & 0 \\ I+2J+K & 2I+J & I & 0 \\ J+2K+L & I+2J+K & 2I+J & I \end{bmatrix}\right) \quad (15)$$

Then each row in each term is treated separately. For example, in the first term the first row can be expressed as:

$$\begin{bmatrix} A & 2A+B & A+2B+C & B+2C+D \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}\cdot\begin{bmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ A & B & C & D \end{bmatrix}\cdot\begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (16)$$

The first matrix in the right-hand-side of (16) is a fixed coefficient matrix like the one described in the previous modes. The transform of such matrix, donated as $[C_{4a}]$ is:

$$C_{4a} = \begin{bmatrix} 0.25 & -0.3162 & 0.25 & -0.158 \\ 0.3162 & -0.4 & 0.3162 & -0.2 \\ 0.25 & -0.3162 & 0.25 & -0.158 \\ 0.158 & -0.2 & 0.158 & -0.1 \end{bmatrix} \quad (17)$$

From (17) we see that the first row (R1) is the same as the third row (R3) thus both form only one adder array and the resulting transform coefficients will be the same. Also, the fourth row (R4) is the shift right operation of the second row (R2).

Therefore, only 2 adder arrays are needed to generate all the transform coefficients result from multiplying $[C_{4a}]$ by the transform of the above block in (16). We donate this sub-term as L_1. Figure 2a and 2b show the butterfly structures generated from the 2 adder arrays.

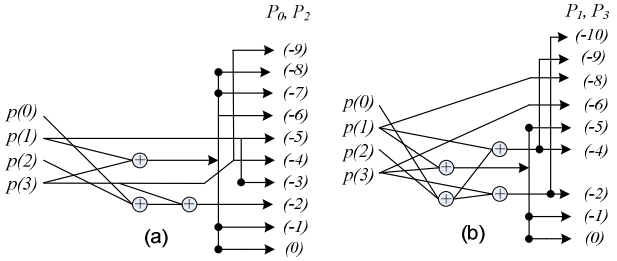

Figure 3. Butterfly structures for Mode 4

Only 3 additions are needed to compute the first and third transform coefficients (P_0,P_2) in each column and 4 additions for the second and fourth (P_1,P_3) as shown in Fig. 3. By further examining the butterfly structures, it can be seen that there is a unique patterns of 2-inputs additions (for example p(0) + p(2) in Fig. 3 appears in both structures). We utilize that to achieve more compression in additions. A total of 20 additions are needed for all the transform coefficients computation of L_1.

Note that the rows in the first term of (15) have a vertical shift relation. Exploiting this relation simplify the computation. For example, L_2 (of the second row) is expressed as:

$$L_2 = C_{4b}\cdot HT\left(\begin{bmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ A & B & C & D \end{bmatrix}\right),\ C_{4b} = \begin{bmatrix} 0.25 & -0.316 & 0.25 & -0.158 \\ 0.158 & -0.2 & 0.158 & -0.1 \\ -0.25 & 0.316 & -0.25 & 0.158 \\ -0.316 & 0.4 & -0.316 & 0.2 \end{bmatrix}$$

$[C_{4b}]$ have the same rows as $[C_{4a}]$ and can be rewritten as:

$$C_{4b} = \begin{bmatrix} R1 \\ R3 \\ -R1 \\ -R2 \end{bmatrix}, \qquad C_{4c} = \begin{bmatrix} R1 \\ -R3 \\ -R1 \\ R2 \end{bmatrix}, \; C_{4d} = \begin{bmatrix} R1 \\ -R2 \\ R1 \\ -R3 \end{bmatrix} \tag{18}$$

Therefore the same adder arrays and the same results from L_1 are used for L_2 and also for the third (L_3) and fourth (L_4) rows with no additional computations. For completeness, C_{4c} and C_{4d} of the third and fourth rows respectively are shown in (18).

Due to page limit we omit the derivation of the other Intra_4x4 modes. Using the same analysis described above, the transform domain intra prediction of these modes is straightforward.

3.2. Intra_16x16

Intra_16x16 supports 4 prediction modes. The transform domain prediction of these modes involves much less computation than Intra_4x4 and can be derived using the same analysis drawn above. Moreover, Prediction samples of the first two modes: 0 (Vertical) and 1 (Horizontal) can be extracted directly from the first two modes of Intra_4x4 and does not require any computations.

4. COMPUTATIONAL COMPLEXITY ANALYSIS

NEDA keeps the data inputs in their natural bit-parallel form and manipulates the fixed coefficients in a distributed bit domain leading to only addition and shift operations and as a result does not require any multiplication or ROM. Furthermore, NEDA also exhibits scalability. When higher precision is desired, modules required to produce the extra DA bit outputs can be added without making any change to the old system.

Predicted blocks are computed on a row by row basis and internal results are shared by other modes. For example, L_1-L_4 of mode 4 appears in other modes and is reused with no additional computations. The compression scheme is incorporated whenever possible to achieve more savings in the addition operations.

Table 1 shows a consistent comparison between the operations required for Intra_4x4 prediction using matrix multiplication proposed in [6] and that is using our method. A maximum of 30% increase in the number of additions is reported for the most complicated modes which are entirely compensated for by eliminating matrix multiplications operations and the need to store pre-computed matrices required by each mode. A small amount of shift operations are required at the final stage of NEDA which is much less expensive than multiplication operations. For example, mode 3 would require only 182 shift operations compared to 168 multiplications. Intra_16x16 results are shown in Table 2 with no increase in additions as most of the computations can be reused from Intra_4x4.

5. CONCLUSION

Video transcoding in the transform domain and multimedia transcoding in wireless and mobile networks has necessities the need for computationally efficient algorithms. Using distributed arithmetic, we presented a basis for a low computation transform domain intra prediction method suitable for complexity/area sensitive systems.

Table 1. Computational Complexity of Intra_4x4 Prediction

Mode	[6]		proposed	
	Multiplication	Add	Multiplication	Add
0	8	12	Zero Multiplication Operation	12
1	8	12		12
2	2	1		1
3	168	136		180
4	232	200		252
5	192	176		164
6	192	176		128
7	128	112		112
8	64	48		48

Table 2. Computational Complexity of Intra_16x16 Prediction

Mode	[6]		proposed	
	Multiplication	Add	Multiplication	Add
0	0	0	Zero Multiplication Operation	0
1	0	0		0
2	8	7		7
3	26	72		72

6. ACKNOWLEDGEMENT

The authors acknowledge the support of the U.S. Department of Energy (DoE), EETAPP program DE97ER12220, the Governor's Information Technology Initiative, and NSF, INF 6-001-006.

7. REFERENCES

[1] I. Ahmad, X. Wei, Y. Sun, Y. Zhang, "Video Transcoding: An Overview of Various Techniques and Research Issues," *IEEE Trans. Multimedia,* vol. 7, p. 12, 2005.

[2] "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC)," 2003.

[3] Y. Hong, K. Lee, J. Kim, and W. Cho, "High Speed Architecture for MPEG-2/H.264 Video Transcoding," in *International Symposium on Communications and Information Technologies*, pp. 674--678, 2006.

[4] V. A. Nguyen and T. Peng, "Efficient video transcoding between H.263 and H.264/AVC standards," in *IEEE International Symposium on Circuits and Systems* pp. 904-907 Vol. 2, 2005.

[5] P. Assuncao and M. Ghanbari, "A Frequency-Domain Video Transcoder for Dynamic Bit-Rate Reduction of MPEG-2 Bit Streams," *IEEE Trans. Circuits Syst. Video Technol.,* vol. 8, p. 15, 1998.

[6] C. Chen, W. Ping-Hao, and H. Chen, "Transform-domain intra prediction for H.264," in *IEEE International Symposium on Circuits and Systems*, pp. 1497-1500 Vol. 2, 2005.

[7] A. M. Shams, A. Chidanandan, W. Pan, and M. A. Bayoumi, "NEDA: a low-power high-performance DCT architecture," *IEEE Transactions on Signal Processing,* vol. 54, pp. 955-964, 2006.

[8] S. A. White, "Applications of distributed arithmetic to digital signal processing: a tutorial review," *IEEE ASSP Magazine,* vol. 6, pp. 4-19, 1989.