Efficient Fast 1-D 8 × 8 Inverse Integer Transform for VC-1 Application

Chih-Peng Fan, Member, IEEE, and Guo-An Su

Abstract—In this paper, one-dimensional (1-D) fast 8 × 8 inverse integer transform algorithm for Windows Media Video 9 (WMV-9/VC-1) is proposed. Based on the symmetric property of the integer transform matrix and the matrix operations, which denote the row/column permutations and the matrix decompositions, the efficient fast 1-D 8 × 8 inverse integer transform is developed. Therefore, the computational complexities of the proposed fast inverse transform are smaller than those of the direct method and the previous fast method. With the low complexity, the proposed fast algorithm is suitable to accelerate the video coding computations.

Index Terms—8 × 8 inverse integer transform, fast algorithm, video coding, WMV9/VC-1.

I. INTRODUCTION

In recent years, there have been a lot of new researches and developments for the video coding, such as H.264/AVC and WMV-9/VC-1 [1]–[5]. The VC-1 standard, which is developed by Microsoft Corporation and standardized by the Society of Motion Picture and Television Engineers (SMPTE), is applied to video encoder, video decoder, and the compressed steaming video data, which are transferred via the internet network [1]. For the implementation of the delivery of the VC-1 system, the whole media system is separated into three parts, which are the authoring, the distribution, and the playback. In the authoring and the playback of the VC-1 system, the codec of the video is used. Thus, the transform schemes of the forward and the inverse in the encoder are required to transform the spatial data of the temporal residuals, which are the differences between the actual block and the prediction block generated by the motion compensation. In the decoder, the inverse integer transformation can be used to transform the de-quantization data, and then the image blocks will be reconstructed. To achieve the advantages of various block sizes [1], the transformation block sizes are one 8 × 8 block, two 8 × 4 blocks with the horizontal arrangement, two 4 × 8 blocks with the vertical arrangement, and four 4 × 4 blocks. Since the complexity of the 1-D 8 × 8 inverse transform is more complex than that of the 1-D 4 × 4 inverse transform, we will focus on the development of the fast 1-D 8 × 8 inverse integer transform for VC-1 in the paper.

In [2], [3], the construction of the fast inverse transform matrix is explained and the implementation of the fast inverse transform can be performed by using 16-bit arithmetic, where the performance of the rate-distortion is similar to the 32-bit or floating point implementation of the DCT. In [6], [7], the authors show that the fast and accurate approximations of DCT, which are related with the integer DCT transform, can be obtained without using any multiplication. The proposed algorithm is called BinDCT. This particular implementation requires 30 additions and 13 shift operations.

The rest of this paper is organized as follows. In Section 2, the review of the inverse integer transforms for VC-1 is described. In Section 3, the three proposed fast 1-D 8 × 8 inverse integer transform algorithms for VC-1 are introduced with the proposed matrix factorizations. Then the computational complexities and bit-exactness issues of three proposed decomposition methods for the 1-D 8 × 8 fast inverse integer transform of VC-1 are discussed. In Section 4, the comparisons of the computational complexities between the proposed fast methods and the fast method in [3] are shown. In Section 5, the bit width analysis for the implementation of the row-column wise method of the 2-D 8 × 8 inverse transforms is discussed. Finally, we give a conclusion.

II. REVIEW OF INVERSE INTEGER TRANSFORM FOR VC-1

From [1], the matrices of the 1-D 8 × 8 and 4 × 4 inverse integer transforms of VC-1 are shown in (1) and (2), respectively.

\[
T_{WMV8}^{\text{WMV8}} = \begin{bmatrix}
12 & 12 & 12 & 12 & 12 & 12 & 12 & 12 \\
16 & 15 & 9 & 4 & -4 & -9 & -15 & -16 \\
16 & 6 & -6 & -16 & -16 & -6 & 6 & 16 \\
15 & -4 & -16 & -9 & 9 & 16 & 4 & -15 \\
12 & -12 & -12 & 12 & 12 & -12 & -12 & 12 \\
9 & -16 & 4 & 15 & -15 & -4 & 16 & -9 \\
6 & -16 & 16 & -6 & -6 & 16 & -16 & 6 \\
4 & -9 & 15 & -16 & 16 & -15 & 9 & -4
\end{bmatrix}
\]

(1)

and

\[
T_{WMV4}^{\text{WMV4}} = \begin{bmatrix}
17 & 17 & 17 & 17 \\
22 & 10 & -10 & -22 \\
17 & -17 & -17 & 17 \\
10 & -22 & 22 & -10
\end{bmatrix}
\]

(2)
In (1) and (2), the elements of the right and the left sides have symmetric properties, which can be applied to decompose the original transform matrix into the product of the sparse matrices further. From [2], the implementation of the 2-D $8 \times 8$ inverse integer transform for VC-1 is achieved by using the row-column wise method as follows.

$$R = T_{WMV8}^T \cdot X \cdot T_{WMV8}/1024,$$

(3)

where the denotation “$T$” is the transpose operation, $X$ is the output of the de-quantization process, and $R$ is the reconstruction matrix. From [2], the two-stage row-column wise scheme is described as follows. The first stage, where the 1-D rows of the de-quantization output are inversely transformed, is shown in the following.

$$D_1 = X \cdot T_{WMV8}/16,$$

(4)

where the matrix $D_1$ is the output of the first stage. For the hardware implementations, which use the cell-based or FPGA design flow, the elements of the de-quantization output $X$ are defined as the fixed point values [8]. In the second stage, the 1-D columns of $D_1$ in (4) are inversely transformed as follows.

$$R = T_{WMV8}^T \cdot D_1/64.$$ 

(5)

III. PROPOSED EFFICIENT FAST 1-D $8 \times 8$ INVERSE INTEGER TRANSFORM FOR VC-1

In the section, three proposed matrix decomposition schemes for the fast 1-D $8 \times 8$ inverse integer transform in VC-1 are discussed, and then the computational complexities of three proposed methods are also explained. The purpose of three proposed matrix decomposition methods is to reduce the computational complexities, which are referred to as the numbers of additions and shift operations. By applying the symmetric property of the transform matrix, the transform matrix in (1) can be rewritten as follows.

$$T_{WMV8} = T_1 \cdot P_1,$$

(6)

where

$$T_1 = \begin{bmatrix}
12 & 12 & 12 & 12 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -4 & -9 & -15 & -16 \\
16 & 6 & -6 & -16 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 9 & 16 & 4 & -15 \\
12 & -12 & -12 & 12 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -15 & -4 & -9 & 16 \\
6 & -16 & 16 & -6 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -15 & 16 & 16 & -9 \\
0 & 0 & 0 & 0 & 16 & -15 & 9 & -4
\end{bmatrix},$$

and

$$P_1 = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}.$$ 

After the matrix decomposition process in (6), the number of the zero-value elements in the transform matrix $T_{WMV8}$ is increased, and the sparse matrices are shown in $T_1$ and $P_1$. The function of $P_1$ is the post-process matrix for the input data of the 1-D $8 \times 8$ inverse integer transform, and the post-process only uses the additions and subtracts. The computational complexities of $P_1$ are 8 additions. In (6), the elements of $T_1$ are scattered, and then the rearrangement of the elements in $T_1$ is required to group the elements of $T_1$ into two $4 \times 4$ independent matrices. By permuting the rows of $T_1$, it is rewritten as follows.

$$T_1 = P_r \cdot \hat{T},$$

(7)

where

$$P_r = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix},$$

and

$$\hat{T} = \begin{bmatrix}
12 & 12 & 12 & 12 & 0 & 0 & 0 & 0 \\
16 & 6 & -6 & -16 & 0 & 0 & 0 & 0 \\
12 & -12 & -12 & 12 & 0 & 0 & 0 & 0 \\
6 & -16 & 16 & -6 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -4 & -9 & -15 & -16 \\
0 & 0 & 0 & 0 & -15 & -4 & 16 & -9 \\
0 & 0 & 0 & 0 & 16 & -15 & 9 & -4
\end{bmatrix}.$$ 

In (7), the implementation of $P_r$ does not need any operations of additions and shifts. In the matrix $\hat{T}$, it can be further expressed as follows.

$$\hat{T} = \hat{T}_{00} \oplus \hat{T}_{11},$$

(8)

where

$$\hat{T}_{00} = \begin{bmatrix}
12 & 12 & 12 & 12 \\
16 & 6 & -6 & -16 \\
12 & -12 & -12 & 12 \\
6 & -16 & 16 & -6
\end{bmatrix},$$

and

$$\hat{T}_{11} = \begin{bmatrix}
-4 & -9 & -15 & -16 \\
9 & 16 & 4 & -15 \\
-15 & -4 & 16 & -9 \\
16 & -15 & 9 & -4
\end{bmatrix}.$$
and the symbol “⊕” is the direct-sum operator.

In (8), the elements in the matrix \( \tilde{T}_{00} \) have the symmetric property, and then the matrix can be further decomposed into the product of the sparse matrices. In the 1-D 8 \( \times \) 8 forward integer transform [3], the computation of \( \tilde{T}_{00} \) can be called the computation of the even frequency part in the transform matrix, and the computation of \( \tilde{T}_{11} \) can be called the computation of the odd frequency part in the transform matrix. In the case of the 1-D 8 \( \times \) 8 inverse integer transform, the matrix of the inverse integer transform is the transpose of the matrix of the forward integer transform [3]. Therefore, the direction of the whole data flow of the 1-D 8 \( \times \) 8 inverse integer transform is the reverse direction of the whole data flow of the 1-D 8 \( \times \) 8 forward integer transform. The even frequency parts are defined as the transform outputs with the even frequency indices, where the frequency indices are 0, 2, 4, and 6. The odd frequency parts are defined as the transform outputs with the odd frequency indices, where the frequency indices are 1, 3, 5, and 7. Generally, the challenge for the development of the integer transform is to design the decomposition method of \( \tilde{T}_{11} \). To clarify the decomposition schemes, three proposed decomposition methods are discussed in three cases as follows.

A. The Proposed Decomposition Scheme : Case 1

The symmetric property of \( \tilde{T}_{00}/4 \) can be applied to factorize the original matrix into the product of the sparse matrices and the decomposition scheme is shown as follows.

\[
\frac{\tilde{T}_{00}}{4} = \begin{bmatrix} 3 & 3 & 3 & 3 \\ 4 & 3/2 & -3/2 & -4 \\ 3 & -3 & 0 & 0 \\ 3/2 & -4 & 4 & -3/2 \end{bmatrix} = R_1 \cdot R_2, \quad (9)
\]

where

\[
R_1 = \begin{bmatrix} 3 & 3 & 0 & 0 \\ 0 & 0 & -3/2 & -4 \\ 3 & -3 & 0 & 0 \\ 0 & 0 & 4 & -3/2 \end{bmatrix}
\]

and

\[
R_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}.
\]

In the elements of \( R_1 \), the computation of the \(-3x/2\) (i.e., \(-x \cdot x/2\)) is replaced with \(-[x + (x \gg 1)]\), where \(x \gg 1\) means \(x\) right shifts 2 bits, and the two right-shift decimal bits are still reserved by widening the bit widths. The computational complexities of \( R_3 \) are 4 additions and 4 shift operations. Then the common factor \(3/4\) in \( R_4\) is also extracted to reduce the computational complexities. The computational complexities of \( R_4 \) are 12 additions and 4 shift operations. Finally, the total computational complexities of \( \tilde{T}_{11}/4 \) in (9) are 12 additions and 6 shift operations. Next, the \( \tilde{T}_{11}/4 \) can be further factorized into the matrix product and the decomposition scheme is shown as follows.

\[
\frac{\tilde{T}_{11}}{4} = \begin{bmatrix} -1 & -9/4 & -15/4 & -4 \\ 9/4 & 4 & 1 & -15/4 \\ 4 & -15/4 & 9/4 & -1 \end{bmatrix} = R_3 \cdot R_4, \quad (10)
\]

In the elements of \( R_4 \), the computation of the \(\pm 3x/4\) is replaced with \(\pm[x - (x \gg 2)]\), where \((x \gg 2)\) means \(x\) right shifts 2 bits, and the two right-shift decimal bits are still reserved by widening the bit widths. The computational complexities of \( R_3 \) are 4 additions and 4 shift operations. Then the common factor \(3/4\) in \( R_4\) is also extracted to reduce the computational complexities. The computational complexities of \( R_4 \) are 12 additions and 4 shift operations. Finally, the total computational complexities of \( \tilde{T}_{11}/4 \) in (10) are 16 additions and 8 shift operations. To consider the bit-exactness issue, the alternate representation of (10) is shown as follows.

\[
(\tilde{T}_{11f} \gg 2) = R_3 \cdot (T_2 \gg 2), \quad (11)
\]

where

\[
T_2 = \begin{bmatrix} -4 & 3 & -3 & 0 \\ -3 & 0 & 4 & -3 \\ 3 & 4 & 0 & -3 \\ 0 & -3 & -3 & -4 \end{bmatrix}
\]

and \( \tilde{T}_{11f} \) is the fixed point representation of \( \tilde{T}_{11} \). By using (6), (7), (8), (9), and (10), the proposed fast 1-D 8 \( \times \) 8 inverse integer transform in Case 1 is given as follows.

\[
T_{WMVS} = 4 \cdot \left( \frac{1}{4} \cdot T_1 \cdot P_1 \right),
\]

\[
= 4 \cdot \left( P_1 \cdot \left( \left[ \frac{1}{4} \cdot \tilde{T}_{00} \right] \oplus \left[ \frac{1}{4} \cdot \tilde{T}_{11} \right] \right) \cdot P_1 \right),
\]

\[
= 4 \cdot \left( P_1 \cdot \left( \left( R_1 \cdot R_2 \right) \oplus \left( R_3 \cdot R_4 \right) \right) \cdot P_1 \right). \quad (12)
\]
In (12), the transform scaling is 4. From the above discussions, the total computational complexities of the proposed fast 1-D $8 \times 8$ inverse integer transform in (12) need 36 additions and 14 shift operations. From (4) and (12), the transform scaling 4 can be absorbed into the de-quantization process, and then the 1-D rows of the de-quantization matrix are inversely transformed as follows.

$$D_1 = X \cdot \{ P_1 \cdot [(R_1 \cdot R_2) \oplus (R_3 \cdot R_4)] \cdot P_1 \} / 16,$$

(13)

where $X$ is the output of the de-quantization with absorbing the transform scaling 4. In (13), the computation of the 1-D $8 \times 8$ inverse integer transform can be performed in the fixed point arithmetic and the alternate representation of (13) can be used as follows.

$$D_{1f} = [X_f \cdot (T_{W MV Sf} \gg 2)] \gg 4,$$

(14)

where $X_f$, $D_{1f}$, and $T_{W MV Sf}$ are the fixed point representations of $X$, $D_1$, and $T_{W MV S}$, respectively.

**B. The Proposed Decomposition Scheme: Case 2**

Similarly, the matrix $T_{00}/8$ can be factorized into the product of the sparse matrices and the decomposition scheme is shown as follows.

$$\frac{T_{00}}{8} = \begin{bmatrix}
\frac{3}{2} & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} \\
\frac{2}{3} & \frac{3}{4} & -\frac{3}{4} & -\frac{2}{3} \\
\frac{3}{4} & -\frac{2}{3} & -\frac{3}{2} & \frac{3}{2} \\
\frac{3}{4} & -\frac{2}{3} & -\frac{3}{2} & -\frac{3}{4}
\end{bmatrix} = R_1 \cdot R_2,$$

(15)

where

$$R_1 = \begin{bmatrix}
\frac{3}{2} & \frac{3}{2} & 0 & 0 \\
\frac{3}{2} & -\frac{3}{2} & 0 & 0 \\
0 & 0 & \frac{3}{4} & -\frac{2}{3} \\
0 & 0 & 2 & -\frac{3}{4}
\end{bmatrix}$$

and

$$R_2 = \begin{bmatrix}
1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 \\
0 & -1 & 1 & 0 \\
-1 & 0 & 0 & 1
\end{bmatrix}.$$

In Case 2, the computational complexities of $R_3$ are 8 additions and 6 shift operations. The computational complexities of $R_2$ are 4 additions. Finally, the total computational complexity of $T_{00}/8$ in (15) is 12 additions and 6 shift operations. Next, the matrix $T_{11}/8$ can be further factorized into the matrix product and the decomposition scheme is shown as follows.

$$\frac{T_{11}}{8} = \begin{bmatrix}
-\frac{1}{2} & -\frac{9}{8} & -\frac{15}{8} & -\frac{2}{3} \\
\frac{9}{8} & 2 & 1/2 & -\frac{15}{8} \\
-\frac{15}{8} & -\frac{1}{2} & 2 & -\frac{9}{8} \\
2 & -\frac{15}{8} & 9/8 & -\frac{1}{2}
\end{bmatrix} = R_3 \cdot R_4,$$

(16)

where

$$R_3 = \begin{bmatrix}
\frac{1}{4} & 0 & 0 & 1 \\
0 & \frac{1}{4} & 1 & 0 \\
0 & 1 & -\frac{1}{4} & 0 \\
-1 & 0 & 0 & 1/4
\end{bmatrix}$$

and

$$R_4 = \begin{bmatrix}
-\frac{2}{3} & \frac{3}{2} & -\frac{3}{2} & 0 \\
-\frac{3}{2} & 0 & 2 & -\frac{3}{2} \\
\frac{3}{2} & 0 & 2 & -\frac{3}{2} \\
0 & -\frac{3}{2} & -\frac{3}{2} & -2
\end{bmatrix}.$$

In (16), the computational complexities of $R_3$ are 4 additions and 4 shift operations. The common factor $3/2$ in $R_4$ is also extracted to reduce the computational complexities. The computational complexities of $R_4$ are 12 additions and 8 shift operations. Finally, the total computational complexities of $T_{11}/8$ in (16) are 16 additions and 12 shift operations. By using the similar bit-exactness concept in (11), the alternate representation of (16) is shown as follows.

$$\frac{T_{11f} \gg 3}{} = \frac{T_3 \gg 2}{} \cdot \frac{T_4 \gg 1}{}.$$

(17)

where

$$T_3 = \begin{bmatrix}
1 & 0 & 0 & 4 \\
0 & 1 & 4 & 0 \\
0 & 4 & -1 & 0 \\
-4 & 0 & 0 & 1
\end{bmatrix},$$

$$T_4 = \begin{bmatrix}
-4 & 3 & -3 & 0 \\
-3 & 0 & 4 & -3 \\
3 & 4 & 0 & -3 \\
0 & -3 & -3 & -4
\end{bmatrix}.$$

By using (6), (7), (8), (15), and (16), the proposed fast 1-D $8 \times 8$ inverse integer transform in Case 2 is given as follows.

$$T_{W MV S} = 8 \cdot \{ P_1 \cdot [(R_1 \cdot R_2) \oplus (R_3 \cdot R_4)] \cdot P_1 \}.$$

(18)

In (18), the transform scaling is 8. From the above discussions, the total computational complexities of the proposed fast 1-D $8 \times 8$ inverse integer transform in (18) need 36 additions and 18 shift operations. From (4) and (18), the transform scaling 8 can be absorbed into the de-quantization process, and then the 1-D rows of the de-quantization matrix are inversely transformed as follows.

$$D_1 = X \cdot \{ P_1 \cdot [(R_1 \cdot R_2) \oplus (R_3 \cdot R_4)] \cdot P_1 \} / 16,$$

(19)

where $X$ is the output of the de-quantization with absorbing the transform scaling 8. In (19), the computation of the 1-D $8 \times 8$ inverse integer transform can be performed in fixed point arithmetic and the alternate representation of (19) can be used as follows.

$$D_{1f} = [X_f \cdot (T_{W MV Sf} \gg 3)] \gg 4,$$

(20)
where $X_f, D_1f,$ and $T_{WMOV8}$ are the fixed point representations of $X, D_1,$ and $T_{WMOV8},$ respectively.

### C. The Proposed Decomposition Scheme : Case 3

In the above Cases 1 and 2, the elements of the decomposed matrices, which are $R_1, R_2, R_3,$ and $R_4,$ include some fractional numbers. In Case 3, the proposed decomposition scheme is to develop the decomposed matrices whose elements are all integer numbers. Similarly, the matrix $\hat{T}_{00}$ can be factorized into the product of the sparse matrices and the decomposition scheme is shown as follows.

$$\hat{T}_{00} = \begin{bmatrix} 12 & 12 & 12 & 12 \\ 16 & 6 & -6 & -16 \\ 12 & -12 & 12 & 12 \\ 6 & -16 & 16 & -6 \end{bmatrix} = R_1 \cdot R_2,$$

where

$$R_1 = \begin{bmatrix} 12 & 12 & 0 & 0 \\ 12 & -12 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad R_2 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & -2 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}.$$

In Case 3, the computational complexities of $R_1$ are 8 additions and 10 shift operations, where the common factor 12 in $R_1$ is also extracted to reduce the computational complexities. The computational complexities of $R_2$ are 4 additions. Finally, the total computational complexities of $\hat{T}_{00}$ in (21) are 12 additions and 18 shift operations. From the method in [3], the decomposition scheme of $\hat{T}_{00}$ in Case 3 is similar to that of the even frequency part in [3]. The differences between the method in [3] and the proposed method in Case 3 are the decomposition method of $\hat{T}_{11},$ which is called the decomposition scheme of the odd frequency part. Next, the matrix $\hat{T}_{11}$ in (8) can be further factorized into the matrix product and the decomposition scheme is shown as follows.

$$\hat{T}_{11} = \begin{bmatrix} -4 & -9 & -15 & -16 \\ 9 & 16 & 4 & -15 \\ -15 & -4 & 16 & -9 \\ 16 & -15 & 9 & -4 \end{bmatrix} = R_3 \cdot R_4,$$  \hspace{1cm} (22)

where

$$R_3 = \begin{bmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 4 & 0 \\ 0 & 4 & -1 & 0 \end{bmatrix} \quad \text{and} \quad R_4 = \begin{bmatrix} -4 & 3 & -3 & 0 \\ -3 & 0 & 4 & -3 \\ 3 & 4 & 0 & -3 \\ 0 & -3 & -3 & -4 \end{bmatrix}.$$

In (22), the computational complexities of $R_3$ are 4 additions and 4 shift operations. The common factor 3 in $R_1$ is also extracted to reduce the computational complexities. Then the computational complexities of $R_4$ are 12 additions and 4 shift operations. Finally, the total computational complexities of $\hat{T}_{11}$ in (22) are 16 additions and 8 shift operations. By using (6), (7), (8), (21), and (22), the proposed fast 1-D $8 \times 8$ inverse integer transform in Case 3 is given as follows.

$$T_{WMOV8} = P_r \cdot [(R_1 \cdot R_2) \oplus (R_3 \cdot R_4)] \cdot P_1,$$  \hspace{1cm} (23)

From the above discussions, the total computational complexities of the proposed fast 1-D $8 \times 8$ inverse integer transform in (23) need 36 additions and 18 shift operations. By (4) and (23), the 1-D rows of the de-quantization matrix are inversely transformed as follows.

$$D_1 = X \cdot \{P_r \cdot [(R_1 \cdot R_2) \oplus (R_3 \cdot R_4)] \cdot P_1\} / 16.$$  \hspace{1cm} (24)

In (24), the computation of the 1-D $8 \times 8$ inverse integer transform can be performed in fixed point arithmetic and the alternate representation can be used as follows.

$$D_1f = (X_f \cdot T_{WMOV8f}) \gg 4.$$  \hspace{1cm} (25)

where $X_f, D_1f,$ and $T_{WMOV8f}$ are the fixed point representations of $X, D_1,$ and $T_{WMOV8},$ respectively. The block and the data flow diagrams of the proposed fast 1-D $8 \times 8$ inverse integer transform are shown in Figs. 1 and 2, respectively.

### IV. COMPARISON AND ANALYSIS OF COMPUTATIONAL COMPLEXITY

First, the two matrix properties, which lead to the numbers of additions and shift operations to be smaller, can be applied to analyze the computational complexities among three proposed decomposition cases and they are listed in the two points. (a)
From Tables I and II, the numbers of the additions and the shift operations of three proposed fast 1-D $8 \times 8$ inverse integer transform algorithms are smaller than those of the direct method. Then the numbers of the additions of three proposed fast 1-D $8 \times 8$ inverse integer transform algorithms are smaller than those of the fast method in [3]. The number of the shift operations of the proposed fast method in Case 1 is smaller than that of the fast method in [3]. If the constant factor of 12 for $x[0]$ and $x[4]$, and the constant factor of 2 for $x[2]$ and $x[6]$ can be pulled into de-quantization process for the fast method in [3], where $x[0]$, $x[2]$, $x[4]$, and $x[6]$ are input data of the inverse integer transform, these result in a lowering of computational complexities by 2 additions and 6 shift operations, which bring it to the total numbers of 36 additions and 12 shift operations, which are slightly better than Case 1. In [3], if only multiplies by powers of 2 are feasible, $x[0]$ and $x[4]$ can be multiplied in hardware by the factor 4 instead, and then the method will lower computational complexities by 4 shifts, which bring it to the total numbers of 38 additions and 14 shift operations.

V. BIT WIDTH ANALYSIS OF 2-D $8 \times 8$ INVERSE INTEGER TRANSFORM FOR VC-1

In this section, the bit width analysis of the implementations of the 2-D $8 \times 8$ inverse integer transforms for three cases is discussed briefly. From (13) in Case 1, the implementation of the first stage of 1-D row-wise transform is shown and the bit width of the first stage output $D_1$ is 20 bits, where the bit width of the de-quantization matrix $X$ is 12 bits, and it is mentioned in [2]. In the second stage for the 1-D column-wise transform, the transform scaling 4 can be absorbed into the post-scaling process of the 1-D row-wise transform. Thus, the columns of $D_1$ in (13) are inversely transformed and they are shown as follows.

$$R = T_{WMV8}^{TF} \cdot D_1/64,$$

$$= 4 \cdot \left\{ P_T \cdot \left[ \left( R_T^0 \cdot R_T^4 \right) \oplus \left( R_T^1 \cdot R_T^5 \right) \right] \cdot P_T^T \right\} \cdot D_1/64,$$

$$= \left\{ P_T \cdot \left[ \left( R_T^0 \cdot R_T^4 \right) \oplus \left( R_T^1 \cdot R_T^5 \right) \right] \cdot P_T^T \right\} \cdot \left( 4 \cdot D_1 \right)/64,$$

$$= \left\{ P_T \cdot \left[ \left( R_T^0 \cdot R_T^4 \right) \oplus \left( R_T^1 \cdot R_T^5 \right) \right] \cdot P_T^T \right\} \cdot D_1/64,$$

(26)

where $D_1^T$ is the row-wise transformed output which absorbs the transform scaling 4. In (26), the bit width of the reconstruction outputs $R$ is 24 bits. The computation of the 1-D column-wise inverse integer transform for Case 1 can be performed in the fixed point arithmetic and the alternate representation of (26) can be used as follows.

$$R_f = \left[ \left( T_{WMV8}^{TF} \gg 2 \right) \cdot D_1^T \right] \gg 6,$$

(27)

where $R_f, D_1^T, and T_{WMV8}^{TF}$ are the fixed point representations of $R$, $D_1^T$, and $T_{WMV8}^{TF}$, respectively. For the implementation of the 2-D $8 \times 8$ inverse integer transform with Case 2, when the bit width of the de-quantization matrix $X$ is 12 bits, the bit width of the output in the first stage in (19) is 20 bits and that of the reconstruction outputs $R$ is 24 bits. For the implementation of the 2-D $8 \times 8$ inverse integer transform with Case 3, when the bit width of the de-quantization matrix $X$ is 12 bits, the output
in the first stage in (24) and the reconstruction outputs \( R \) are 20 bits and 24 bits, respectively.

VI. CONCLUSION

The 1-D 8 \( \times \) 8 fast inverse integer transform algorithm for VC-1 is proposed in this paper. Based on the matrix symmetric property and matrix decompositions, the efficient fast 1-D 8 \( \times \) 8 inverse integer transform is developed. The computational complexities of the proposed fast inverse integer transform are smaller than those of the fast method in [3]. By computer simulations, the proposed fast 8 \( \times \) 8 inverse integer transform with Case 1 can save the computational time of the inverse transform about 23\% more than the direct method.

ACKNOWLEDGMENT

This work is supported in part by the National Science Council, Taiwan, R.O.C., under Grant NSC96-2220-E-005-006. The authors would like to thank the anonymous reviewers whose careful reviews and detailed comments help to improve the readability of this paper.

REFERENCES