Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 258613, 14 pages
http://dx.doi.org/10.1155/2015/258613

Research Article

A Systematic Hardware Sharing Method for Unified Architecture Design of H.264 Transforms

Po-Hung Chen,[1] Hung-Ming Chen,[2] and Ing-Chao Lin[3]

[1]*Department of Electronic Engineering, National Formosa University, No. 64, Wenhua Road, Huwei Township, Yunlin County 632, Taiwan*
[2]*Department of Computer Science and Information Engineering, National Taichung University of Science and Technology, No. 129, Section 3, Sanmin Road, North District, Taichung City 404, Taiwan*
[3]*Department of Computer Science and Information Engineering, National Cheng Kung University, No. 1, University Road, Tainan City 701, Taiwan*

Correspondence should be addressed to Po-Hung Chen; paul@nfu.edu.tw

Received 16 July 2014; Accepted 7 September 2014

Academic Editor: Stephen D. Prior

Multitransform techniques have been widely used in modern video coding and have better compression efficiency than the single transform technique that is used conventionally. However, every transform needs a corresponding hardware implementation, which results in a high hardware cost for multiple transforms. A novel method that includes a five-step operation sharing synthesis and architecture-unification techniques is proposed to systematically share the hardware and reduce the cost of multitransform coding. In order to demonstrate the effectiveness of the method, a unified architecture is designed using the method for all of the six transforms involved in the H.264 video codec: 2D 4×4 forward and inverse integer transforms, 2D 4×4 and 2×2 Hadamard transforms, and 1D 8×8 forward and inverse integer transforms. Firstly, the six H.264 transform architectures are designed at a low cost using the proposed five-step operation sharing synthesis technique. Secondly, the proposed architecture-unification technique further unifies these six transform architectures into a low cost hardware-unified architecture. The unified architecture requires only 28 adders, 16 subtractors, 40 shifters, and a proposed mux-based routing network, and the gate count is only 16308. The unified architecture processes 8 pixels/clock-cycle, up to 275 MHz, which is equal to 707 Full-HD 1080 p frames/second.

1. Introduction

Video coding standards commonly use transform coding techniques—discrete cosine transforms (DCTs) are widely used in image and video compression standards, such as JPEG [1], MPEG-1/2 [2, 3], and MPEG-4 [4]. Unlike the DCTs used in previous standards, H.264 [5] uses integer transform matrices for coding, so there is no mismatch between the forward and inverse transforms [6, 7] and the complexity is significantly less than that for a DCT. H.264 also provides a specific transform for each prediction mode, and blocks of size 16×16 down to 4×4 pixels can be used for motion prediction. The prediction modes are organized in a tree-structured manner, which allows flexible combination of different motion compensation block sizes inside a 16×16-pixel macroblock. Therefore, H.264 achieves better compression, but it also requires an enormous number of computations.

H.264 requires the computation of three transforms, 8×8, 4×4 integer transforms, and 4×4 Hadamard transforms used in the luma component, and two transforms, 4×4 integer transforms and 2×2 Hadamard transforms, for the chroma components. Every transform of H.264 needs a corresponding hardware implementation, which results in a high hardware cost for all of the transforms involved. In recent years, some transform architectures for H.264 encoder/decoder that reduce the hardware cost have been proposed. In general, for low cost multiple transforms of video codecs, hardware sharing is the most suitable technique [8–26]. In [8–13], all of the 4×4 transforms are realized as a shared architecture. In [14], 2D 4×4 and 2×2 transforms are implemented in a FPGA and integrated in a multicore

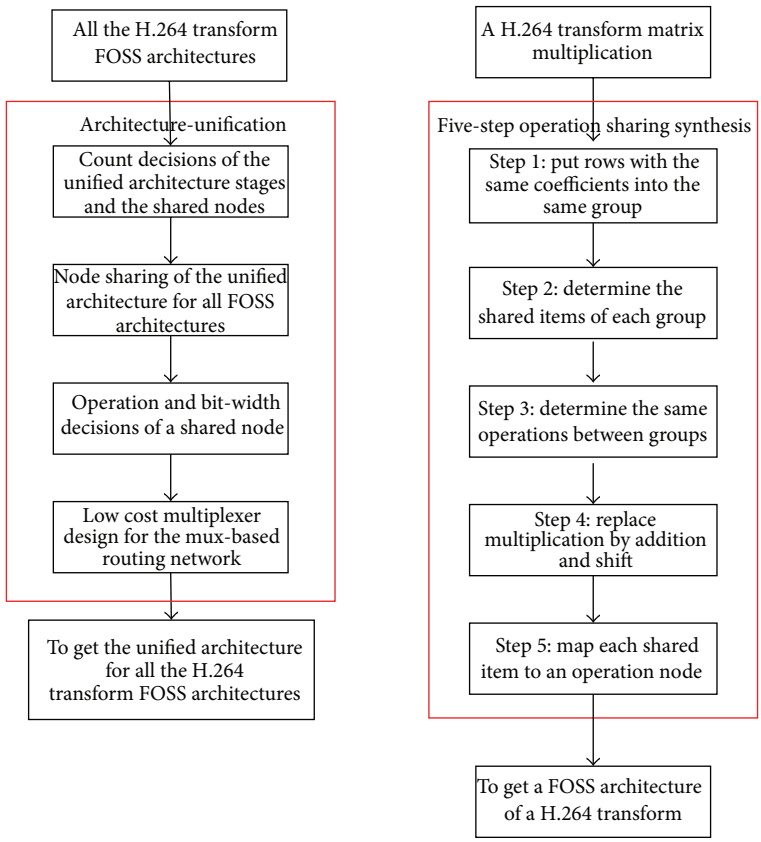

Figure 1: The proposed hardware sharing method.

embedded system. In [15, 16], pipeline shared architectures for the 8 × 8 forward/inverse integer transform are demonstrated. For decoder use only, [17, 18] demonstrate transform processors for 8 × 8 and 4 × 4 inverse integer transforms and a 4 × 4 Hadamard transform. A 2 × 2 Hadamard transform has even been embedded into a shared architecture in [19, 20]. In [21–23], inverse transforms for a multistandard decoder are proposed. In [24], a cost-sharing architecture for an 8 × 8 integer cosine transform is proposed, which supports multiple video encoders. In [25], a unique kernel for multistandard video encoder transforms is presented and a configurable butterfly array (CBA) is also proposed, which supports both the forward transform and the inverse transform in the unified architecture of the multistandard video encoder in [26].

Although these studies demonstrate combined architectures for multiple transforms, a single architecture has not been designed for the whole set of forward and inverse transforms for H.264 encoder and decoder. Therefore, this study designs a unified architecture for the complete transform functionality of a H.264 codec, while still maintaining the low cost and high speed characteristics. In addition, sharing the hardware for the same operations reduces the hardware cost, and these studies use a hardware sharing technique to reduce hardware cost. However, no systematic hardware sharing method has been proposed in the literatures. This paper proposed a novel method that includes a five-step operation sharing synthesis (FOSS) and architecture-unification techniques, to systematically share the hardware and reduce the cost of multitransform coding.

In order to design the low cost unified architecture, a hardware sharing method is proposed, as shown in Figure 1. Firstly, six transform architectures are designed for low cost, using the proposed five-step operation sharing synthesis (FOSS) technique. Secondly, these low cost FOSS architectures are merged into a single architecture, using the proposed architecture-unification technique. The details of these techniques are described in the later sections. Section 2 describes the FOSS architecture design that reduces the hardware cost for each H.264 transform. Section 3 demonstrates the unification of all the low cost transform FOSS architectures into a single architecture, to eliminate the redundant hardware. The complexity and performance of the unified architecture are analyzed in Section 4, and Section 5 concludes the paper.

2. The Five-Step Operation Sharing Synthesis Technique

This paper firstly describes a five-step operation sharing synthesis (FOSS) technique and demonstrates the effectiveness of this technique by building low cost 1D 8 × 8 inverse integer transform and 2D 4 × 4 transform architectures. The procedure for the FOSS technique is as follows.

Step 1. Put rows with the same coefficients into the same group.

Step 2. Determine the same operations in each group.

Step 3. Determine the same operations between groups.

Step 4. Replace multiplication by addition and shift.

Step 5. Map each shared item to an operation node.

The basic idea is to systematically synthesize an architecture that shares all of the same operations in a matrix multiplication to reduce the cost of hardware.

2.1. FOSS Architecture for a 1D 8 × 8 Inverse Integer Transform. Taking 1D 8 × 8 inverse transform as an example, a low cost architecture, called FOSS architecture, is designed using the proposed FOSS technique. The 1D 8 × 8 inverse integer transform is defined as

$$Z = E_i X E_i^T, \tag{1}$$

where

$$E_i = \begin{bmatrix} 1 & 1.5 & 1 & 1.25 & 1 & 0.75 & 0.5 & 0.375 \\ 1 & 1.25 & 0.5 & -0.375 & -1 & -1.5 & -1 & -0.75 \\ 1 & 0.75 & -0.5 & -1.5 & -1 & 0.375 & 1 & 1.25 \\ 1 & 0.375 & -1 & -0.75 & 1 & 1.25 & -0.5 & -1.5 \\ 1 & -0.375 & -1 & 0.75 & 1 & -1.25 & -0.5 & 1.5 \\ 1 & -0.75 & -0.5 & 1.5 & -1 & -0.375 & 1 & -1.25 \\ 1 & -1.25 & 0.5 & 0.375 & -1 & 1.5 & -1 & 0.75 \\ 1 & -1.5 & 1 & -1.25 & 1 & -0.75 & 0.5 & -0.375 \end{bmatrix}. \tag{2}$$

E_i is an 8 × 8 inverse integer transform matrix and X is an 8 × 8 pixel block. E_i^T is the transpose matrix of E_i and Z is an 8 × 8 matrix output of a 2D inverse integer transform. Since the 2D transform is separable, by using the column-row decomposition method, the computation can be converted to 1D row transform followed by a 1D column transform. These operations can be represented as follows:

$$Z = E_i X E_i^T \Longrightarrow Y = X E_i^T \Longrightarrow Z = E_i Y. \tag{3}$$

Therefore, a 2D transform can be calculated using two steps. The first step is a 1D transform $Y = XE_i^T$ and the second step is the other 1D transform $Z = E_i Y$. The first row of X, $X_0 \sim X_7$, multiplied by the inverse transform matrix equals the first row of Y, $Y_0 \sim Y_7$, as shown in the following:

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \\ Y_6 \\ Y_7 \end{bmatrix} = \begin{bmatrix} 1 & 1.5 & 1 & 1.25 & 1 & 0.75 & 0.5 & 0.375 \\ 1 & 1.25 & 0.5 & -0.375 & -1 & -1.5 & -1 & -0.75 \\ 1 & 0.75 & -0.5 & -1.5 & -1 & 0.375 & 1 & 1.25 \\ 1 & 0.375 & -1 & -0.75 & 1 & 1.25 & -0.5 & -1.5 \\ 1 & -0.375 & -1 & 0.75 & 1 & -1.25 & -0.5 & 1.5 \\ 1 & -0.75 & -0.5 & 1.5 & -1 & -0.375 & 1 & -1.25 \\ 1 & -1.25 & 0.5 & 0.375 & -1 & 1.5 & -1 & 0.75 \\ 1 & -1.5 & 1 & -1.25 & 1 & -0.75 & 0.5 & -0.375 \end{bmatrix} \times \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix}. \tag{4}$$

The matrix multiplication in (4) requires 64 multiplications and 56 additions. This paper proposes a novel operation sharing synthesis technique to reduce the hardware cost, and the effectiveness of this technique is demonstrated by applying this technique to a 1D 8×8 inverse integer transform.

Step 1. Put rows with the same coefficients into the same group, to determine the same operations in the next step more easily:

$$\begin{aligned} Y_0 &= X_0 + 1.5X_1 + X_2 + 1.25X_3 + X_4 + 0.75X_5 + 0.5X_6 + 0.375X_7 \\ Y_7 &= X_0 - 1.5X_1 + X_2 - 1.25X_3 + X_4 - 0.75X_5 + 0.5X_6 - 0.375X_7 \end{aligned}$$

$$\begin{aligned} Y_3 &= X_0 + 0.375X_1 - X_2 - 0.75X_3 + X_4 + 1.25X_5 - 0.5X_6 - 1.5X_7 \\ Y_4 &= X_0 - 0.375X_1 - X_2 - 0.75X_3 + X_4 - 1.25X_5 - 0.5X_6 + 1.5X_7 \end{aligned}$$

$$\begin{aligned} Y_1 &= X_0 + 1.25X_1 + 0.5X_2 - 0.375X_3 - X_4 - 1.5X_5 - X_6 - 0.75X_7 \\ Y_6 &= X_0 - 1.25X_1 + 0.5X_2 + 0.375X_3 - X_4 + 1.5X_5 - X_6 + 0.75X_7 \end{aligned}$$

$$\begin{aligned} Y_2 &= X_0 + 0.75X_1 - 0.5X_2 - 1.5X_3 - X_4 + 0.375X_5 + X_6 + 1.25X_7 \\ Y_5 &= X_0 - 0.75X_1 - 0.5X_2 + 1.5X_3 - X_4 - 0.375X_5 + X_6 - 1.25X_7 \end{aligned}$$

Step 2. Determine the same operations in each group and mark the shared operations using "()," as shown in the following:

$$Y_0 = (X_0 + X_2 + X_4 + 0.5X_6) + (1.5X_1 + 1.25X_3 + 0.75X_5 + 0.375X_7)$$
$$Y_7 = (X_0 + X_2 + X_4 + 0.5X_6) - (1.5X_1 + 1.25X_3 + 0.75X_5 + 0.375X_7)$$

$$Y_3 = (X_0 - X_2 + X_4 - 0.5X_6) + (0.375X_1 - 0.75X_3 + 1.25X_5 - 1.5X_7)$$
$$Y_4 = (X_0 - X_2 + X_4 - 0.5X_6) - (0.375X_1 - 0.75X_3 + 1.25X_5 - 1.5X_7)$$

$$Y_1 = (X_0 + 0.5X_2 - X_4 - X_6) + (1.25X_1 - 0.375X_3 - 1.5X_5 - 0.75X_7)$$
$$Y_6 = (X_0 + 0.5X_2 - X_4 - X_6) - (1.25X_1 - 0.375X_3 - 1.5X_5 - 0.75X_7)$$

$$Y_2 = (X_0 - 0.5X_2 - X_4 + X_6) + (0.75X_1 - 1.5X_3 + 0.375X_5 + 1.25X_7)$$
$$Y_5 = (X_0 - 0.5X_2 - X_4 + X_6) - (0.75X_1 - 1.5X_3 + 0.375X_5 + 1.25X_7)$$

Step 3. Determine the same operations between groups and mark the shared operations using "()," as shown in the following:

$$Y_0 = (X_0 + X_4) + (X_2 + 0.5X_6) + (1.5X_1 + 0.375X_7 + 1.25X_3 + 0.75X_5)$$
$$Y_7 = (X_0 + X_4) + (X_2 + 0.5X_6) - (1.5X_1 + 0.375X_7 + 1.25X_3 + 0.75X_5)$$
$$Y_3 = (X_0 + X_4) - (X_2 + 0.5X_6) + (0.375X_1 - 1.5X_7 - 0.75X_3 + 1.25X_5)$$
$$Y_4 = (X_0 + X_4) - (X_2 + 0.5X_6) - (0.375X_1 - 1.5X_7 - 0.75X_3 + 1.25X_5)$$

$$Y_1 = (X_0 - X_4) - (X_6 - 0.5X_2) + (1.25X_1 - 0.75X_7 - 1.5X_5 - 0.375X_3)$$
$$Y_6 = (X_0 - X_4) - (X_6 - 0.5X_2) - (1.25X_1 - 0.75X_7 - 1.5X_5 - 0.375X_3)$$
$$Y_2 = (X_0 - X_4) + (X_6 - 0.5X_2) + (0.75X_1 + 1.25X_7 - 1.5X_3 + 0.375X_5)$$
$$Y_5 = (X_0 - X_4) + (X_6 - 0.5X_2) - (0.75X_1 + 1.25X_7 - 1.5X_3 + 0.375X_5)$$

Step 4. Replace multiplication by addition and shift. If the coefficient is a second power, shift replaces multiplication. For the other coefficients, addition, subtraction, and shift are all needed to replace multiplication. The shared

operations are indicated using "()," as shown in the following:

$$Y_0 = (X_0 + X_4) + (X_2 + 0.5X_6) + (X_1 + 0.5X_1) + (0.25(X_7 + 0.5X_7)) + (0.25X_3 + X_3) + (X_5 - 0.25X_5)$$

$$Y_7 = (X_0 + X_4) + (X_2 + 0.5X_6) - [(X_1 + 0.5X_1) + (0.25(X_7 + 0.5X_7)) + (0.25X_3 + X_3) + (X_5 - 0.25X_5)]$$

$$Y_3 = (X_0 + X_4) - (X_2 + 0.5X_6) + (0.25(X_1 + 0.5X_1)) - (X_7 + 0.5X_7) + (0.25X_3 - X_3) + (X_5 + 0.25X_5)$$

$$Y_4 = (X_0 + X_4) - (X_2 + 0.5X_6) - [(0.25(X_1 + 0.5X_1)) - (X_7 + 0.5X_7) + (0.25X_3 - X_3) + (X_5 + 0.25X_5)]$$

$$Y_1 = (X_0 - X_4) - (X_6 - 0.5X_2) + (X_1 + 0.25X_1) + (0.25X_7 - X_7) - (X_5 + 0.5X_5) + (0.25(X_3 + 0.5X_3))$$

$$Y_6 = (X_0 - X_4) - (X_6 - 0.5X_2) - [(X_1 + 0.25X_1) + (0.25X_7 - X_7) - (X_5 + 0.5X_5) + (0.25(X_3 + 0.5X_3))]$$

$$Y_2 = (X - X_4) + (X_6 - 0.5X_2) + (X_1 - 0.25X_1) + (0.25X_7 + X_7) - (X_3 + 0.5X_3) - (0.25(X_5 + 0.5X_5))$$

$$Y_5 = (X_0 - X_4) + (X_6 - 0.5X_2) - [(X_1 - 0.25X_1) + (0.25X_7 + X_7) - (X_3 + 0.5X_3) - (0.25(X_5 + 0.5X_5))]$$

The computation complexity for a 1D 8×8 inverse transform is reduced from 64 multiplications and 56 additions in (4) to 24 additions, 16 subtractions, and 18 shift operations in (4), which can be directly mapped to a low cost hardware architecture.

Step 5. Map each shared item to an operation node, where $X_0 \sim X_7$ and $Y_0 \sim Y_7$ are the inputs and outputs of the architecture, respectively, and $A_0 \sim A_6$, $B_0 \sim B_{11}$, and $C_0 \sim C_{11}$ represent the shared nodes. Only four stages are required for the architecture, from input to output. Nodes $A_0 \sim A_{11}$, $B_0 \sim B_{11}$, $C_0 \sim C_7$, and $Y_0 \sim Y_7$ are in the 1st, 2nd, 3rd, and 4th stages, respectively. The operations for each stage of the 1D 8×8 inverse integer transform are summarized as follows.

Stage 1. Consider

$$\begin{aligned} A_0 &= X_1 + (X_1 \gg 1), & A_1 &= X_7 + (X_7 \gg 1), \\ A_2 &= X_1 + (X_1 \gg 2), & A_3 &= X_1 - (X_1 \gg 2), \\ A_4 &= (X_7 \gg 2) - X_7, & A_5 &= (X_7 \gg 2) + X_7, \\ A_6 &= (X_3 \gg 2) + X_3, & A_7 &= (X_3 \gg 2) - X_3, \\ A_8 &= X_5 - (X_5 \gg 2), & A_9 &= X_5 + (X_5 \gg 2), \\ A_{10} &= X_3 + (X_3 \gg 1), & A_{11} &= X_5 + (X_5 \gg 1). \end{aligned} \tag{5}$$

Stage 2. Consider

$$\begin{aligned} B_0 &= X_0 + X_4, & B_1 &= X_2 + (X_6 \gg 1), \\ B_2 &= X_0 - X_4, & B_3 &= X_6 - (X_2 \gg 1), \\ B_4 &= A_0 + (A_1 \gg 2), & B_5 &= (A_0 \gg 2) - A_1, \\ B_6 &= A_2 + A_4, & B_7 &= A_3 + A_5, \\ B_8 &= A_6 + A_8, & B_9 &= A_7 + A_9, \\ B_{10} &= A_{11} + (A_{10} \gg 2), & B_{11} &= A_{10} - (A_{11} \gg 2). \end{aligned} \tag{6}$$

Stage 3. Consider

$$\begin{aligned} C_0 &= B_0 + B_1, & C_1 &= B_0 - B_1, \\ C_2 &= B_2 - B_3, & C_3 &= B_2 + B_3, \\ C_4 &= B_4 + B_8, & C_5 &= B_5 + B_9, \\ C_6 &= B_6 - B_{10}, & C_7 &= B_7 - B_{11}. \end{aligned} \tag{7}$$

Stage 4. Consider

$$\begin{aligned} Y_0 &= C_0 + C_4, & Y_1 &= C_2 + C_6, \\ Y_2 &= C_3 + C_7, & Y_3 &= C_1 + C_5, \\ Y_4 &= C_1 - C_5, & Y_5 &= C_3 - C_7, \\ Y_6 &= C_2 - C_6, & Y_7 &= C_0 - C_4. \end{aligned} \tag{8}$$

Using these operations for the four stages, the FOSS architecture for a 1D 8×8 inverse integer transform is obtained as shown in Figure 2. In Figure 2, the "+" sign on the node represents an addition and the node with the "−" sign represents a subtraction. An arrow represents the data flow. If an input coefficient or a coefficient of the arrows is a second power, a shift operationis used. Note that the FOSS

architecture for a 1D 8×8 inverse integer transform processes 8 input pixels and outputs 8 transformed data in parallel.

2.2. The FOSS Architecture for a 2D 4 × 4 Inverse Integer Transform. Take 2D inverse integer transform as the other example to demonstrate more clearly the proposed FOSS technique. The H.264 4 × 4 inverse integer transform is defined as

$$Y = C_i X C_i^T, \tag{9}$$

where

$$C_i = \begin{bmatrix} 1 & 1 & 1 & 0.5 \\ 1 & 0.5 & -1 & -1 \\ 1 & -0.5 & -1 & 1 \\ 1 & -1 & 1 & -0.5 \end{bmatrix}. \tag{10}$$

Although 2D transforms are separable, the column-row decomposition method is not used in this study. Instead, a direct 2D transform method is used to eliminate the use of a transpose register array, in order to reduce the latency. Firstly, C_i in (9) is replaced by (10), to produce the following 16 × 16 transform matrix:

$$\begin{bmatrix} Y_{00} \\ Y_{01} \\ Y_{02} \\ Y_{03} \\ Y_{10} \\ Y_{11} \\ Y_{12} \\ Y_{13} \\ Y_{20} \\ Y_{21} \\ Y_{22} \\ Y_{23} \\ Y_{30} \\ Y_{31} \\ Y_{32} \\ Y_{33} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0.5 & 1 & 1 & 1 & 0.5 & 1 & 1 & 1 & 0.5 & 0.5 & 0.5 & 0.5 & 0.25 \\ 1 & 0.5 & -1 & -1 & 1 & 0.5 & -1 & -1 & 1 & 0.5 & -1 & -1 & 0.5 & 0.25 & -0.5 & -0.5 \\ 1 & -0.5 & -1 & 1 & 1 & -0.5 & -1 & 1 & 1 & -0.5 & -1 & 1 & 0.5 & -0.25 & -0.5 & 0.5 \\ 1 & -1 & 1 & -0.5 & 1 & -1 & 1 & -0.5 & 1 & -1 & 1 & -0.5 & 0.5 & -0.5 & 0.5 & -0.25 \\ 1 & 1 & 1 & 0.5 & 0.5 & 0.5 & 0.5 & 0.25 & -1 & -1 & -1 & -0.5 & -1 & -1 & -1 & -0.5 \\ 1 & 0.5 & -1 & -1 & 0.5 & 0.25 & -0.5 & -0.5 & -1 & -0.5 & 1 & 1 & -1 & -0.5 & 1 & 1 \\ 1 & -0.5 & -1 & 1 & 0.5 & -0.25 & -0.5 & 0.5 & -1 & 0.5 & 1 & -1 & -1 & 0.5 & 1 & -1 \\ 1 & -1 & 1 & -0.5 & 0.5 & -0.5 & 0.5 & -0.25 & -1 & 1 & -1 & 0.5 & -1 & 1 & -1 & 0.5 \\ 1 & 1 & 1 & 0.5 & -0.5 & -0.5 & -0.5 & -0.25 & -1 & -1 & -1 & -0.5 & 1 & 1 & 1 & 0.5 \\ 1 & 0.5 & -1 & -1 & -0.5 & -0.25 & 0.5 & 0.5 & -1 & -0.5 & 1 & 1 & 1 & 0.5 & -1 & -1 \\ 1 & -0.5 & -1 & 1 & -0.5 & 0.25 & 0.5 & -0.5 & -1 & 0.5 & 1 & -1 & 1 & -0.5 & -1 & 1 \\ 1 & -1 & 1 & -0.5 & -0.5 & 0.5 & -0.5 & 0.25 & -1 & 1 & -1 & 0.5 & 1 & -1 & 1 & -0.5 \\ 1 & 1 & 1 & 0.5 & -1 & -1 & -1 & -0.5 & 1 & 1 & 1 & 0.5 & -0.5 & -0.5 & -0.5 & -0.25 \\ 1 & 0.5 & -1 & -1 & -1 & -0.5 & 1 & 1 & 1 & 0.5 & -1 & -1 & -0.5 & -0.25 & 0.5 & 0.5 \\ 1 & -0.5 & -1 & 1 & -1 & 0.5 & 1 & -1 & 1 & -0.5 & -1 & 1 & -0.5 & 0.25 & 0.5 & -0.5 \\ 1 & -1 & 1 & -0.5 & -1 & 1 & -1 & 0.5 & 1 & -1 & 1 & -0.5 & -0.5 & 0.5 & -0.5 & 0.25 \end{bmatrix} \begin{bmatrix} X_{00} \\ X_{01} \\ X_{02} \\ X_{03} \\ X_{10} \\ X_{11} \\ X_{12} \\ X_{13} \\ X_{20} \\ X_{21} \\ X_{22} \\ X_{23} \\ X_{30} \\ X_{31} \\ X_{32} \\ X_{33} \end{bmatrix}. \tag{11}$$

Secondly, the FOSS technique is used for (11) to implement a 2D 4 × 4 inverse integer transform as shown in Figure 3.

The inputs for the FOSS architecture are $X_{00} \sim X_{33}$. The outputs are $Y_{00} \sim Y_{03}$ and $Y_{30} \sim Y_{33}$ as stage 1 performs additions, and the outputs are $Y_{10} \sim Y_{13}$, $Y_{20} \sim Y_{23}$ when stage 1 performs subtractions.

Similarly, the FOSS architectures for a 2D 2×2 Hadamard transform, a 2D 4×4 forward transform, a 2D 4×4 Hadamard transform, and a 1D 8 × 8 forward transform are designed in the same procedure.

3. Architecture-Unification Technique

When all six FOSS architectures of H.264 transforms have been obtained, the proposed architecture-unification procedure described in the following is then used to construct a shared architecture for the FOSS architectures.

3.1. Count Decisions of the Unified Architecture Stages and the Shared Nodes. The unified architecture must have the largest stage count of all of the FOSS architectures, because every stage in each FOSS architecture must correspond to a stage in the unified architecture (e.g., Figure 4). Figures 4(a) and 4(b) show FOSS architectures with four and three stages, respectively. The stage count for a unified architecture must be the largest stage count of the two FOSS architectures, if the two FOSS architectures are to be unified. Figure 4(a) has four stages and Figure 4(b) has three stages, so the unified architecture must have four stages, as shown in Figure 4(c).

The stage counts for all FOSS architectures are not identical. In order to unify FOSS architectures with different stage counts, every FOSS architecture shares operation nodes from stage one of the unified architecture. For example, Figure 4(a) shows a four-stage FOSS architecture, where the shared stages are from stage one to stage four and Figure 4(b) is a three-stage FOSS architecture, where the shared stages are from stage one to stage three, as shown in Figure 4(c).

In the unified architecture, the count for the shared nodes in stage x must be the maximum count for the operation nodes of stage x among all of the FOSS architectures, as shown in Figure 5. In Figure 5(a), stage one has four nodes and stage two has three nodes, and in Figure 5(b), stage one has three nodes and stage two has four nodes. Therefore, both stage one and stage two have four nodes in the unified architecture, as shown in Figure 5(c).

3.2. Node Sharing in the Unified Architecture for All of the FOSS Architectures. If N transforms are to be unified, one of the two inputs of a node in the unified architecture can have up to N different input paths. In order to reduce the multiplexer

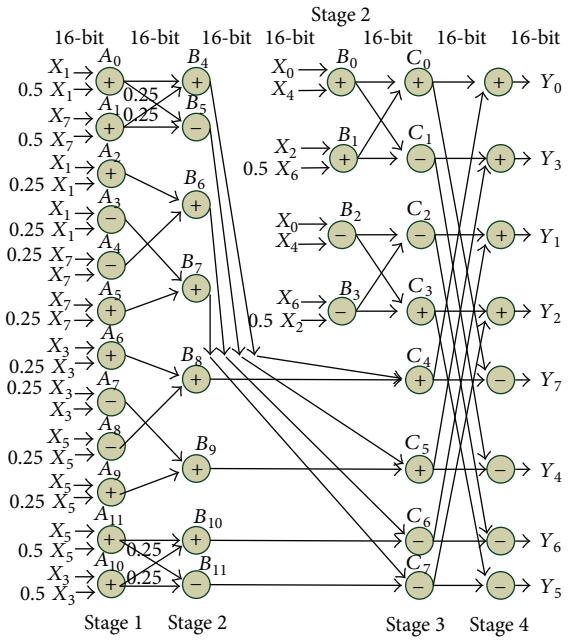

FIGURE 2: A 1D 8 × 8 inverse integer transform.

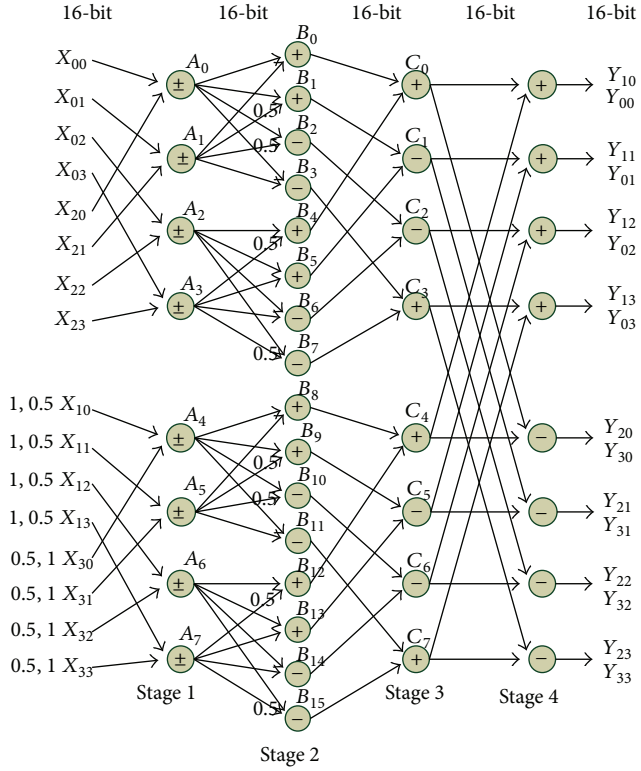

FIGURE 3: A 2D 4 × 4 inverse integer transform.

overhead for input path switching, the number of input paths must be minimized. In order to achieve this goal, the nodes of the unified architecture are shared stage by stage, and every node of a stage is compared for all architectures, using the following procedure.

Step 1. If the nodes have two same input paths and the same operation, go to Step 8.

Step 2. If the nodes have two same input paths but different operations, go to Step 8.

Step 3. If the nodes have only one same input path and the same operation, go to Step 8.

Step 4. If the nodes have only one same input path but different operations, go to Step 8.

Step 5. If the nodes have one or two same input paths with opposite position and their operations are both addition, go to Step 8.

Step 6. If the nodes have not the same input path and have the same operation, go to Step 8.

Step 7. If the nodes have two different input paths and different operations, go to Step 8.

Step 8. The nodes share a node of the unified architecture.

Figure 6 shows an example of how a node is shared, for the 4 FOSS architectures. Figure 6(a) shows the input paths and the operation nodes for a stage for the 4 FOSS architectures. The transform, T_1, shown in Figure 6(a) is used to label each node, from top to bottom ($A_0 \sim A_3$), as shown in Figure 6(b). The nodes of the 4 FOSS architectures ($T_1 \sim T_4$) share a node, using this procedure, and the nodes that share a node use the same label, as shown in Figure 6(b), which is also the number of the shared nodes in the unified architecture, as shown in Figure 6(c). Note that if any one of the inputs of a node in Figure 6(c) has multiple input paths for different transform modes, a multiplexer is required, to select a corresponding input path.

3.3. Operation and the Bit-Width Decisions for a Shared Node. In order to perform the operations of the correspondent nodes for all FOSS architectures, it is necessary to determine the operation for each shared node in the unified architecture.

If the operations for the nodes of the FOSS architectures that share a node are all additions or all subtractions, the operation for this shared node of the unified architecture is addition or subtraction, respectively. If some operations for the nodes of the FOSS architectures that share a node are additions and some operations are subtractions, the operations for this shared node are both subtraction and addition. As shown in Figure 7, C_0 nodes are all adders in all of the FOSS architectures, so a C_0 node is an adder in the unified architecture. Similarly, C_2 is a subtractor. Because the inputs for the correspondent operation nodes in different FOSS architectures are seldom shifted with the same bits, it is not efficient to share a shift operation in the unified architecture.

In order to determine bit-width for a shared node of the unified architecture, bit-widths of the nodes that share a node must be determined first. In order to determine the bit-width for a node of a FOSS architecture, both input and output bit-widths for the FOSS architecture must be

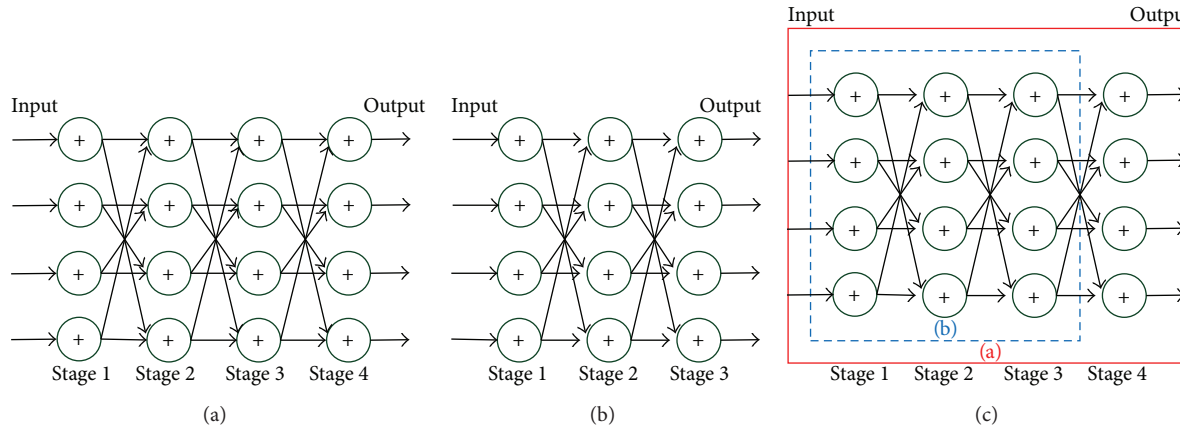

FIGURE 4: Shared stages in the unified architecture: (a) a four-stage FOSS architecture, (b) a three-stage FOSS architecture, and (c) a unified architecture for FOSS architectures (a) and (b), in which the first three stages are shared.

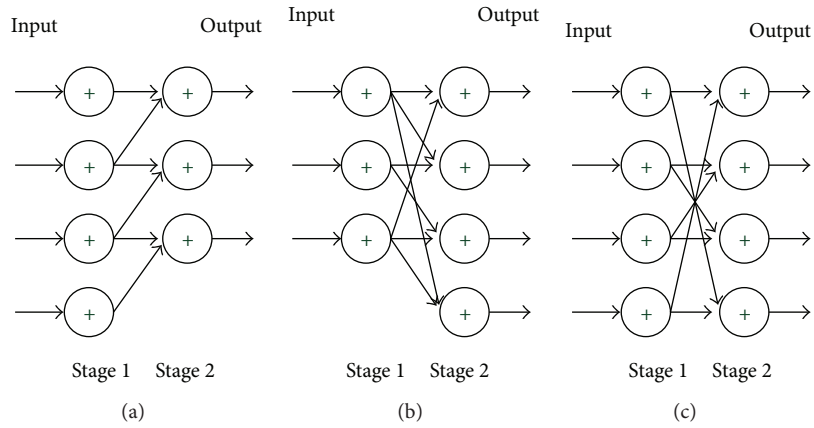

FIGURE 5: Shared node counts for the unified architecture: (a) a FOSS architecture with 4 nodes in stage 1 and 3 nodes in stage 2, (b) a FOSS architecture with 3 nodes in stage 1 and 4 nodes in stage 2, and (c) a unified architecture with 4 nodes in stage 1 and stage 2.

checked in the video standard specification, as shown in Table 1. The input bit-width for a node in the current stage is then determined according to the output dynamic range that results from addition, subtraction, and the shift of the nodes in the previous stage. Note that the dynamic range accumulates stage by stage from input to output, for a FOSS architecture. For the unified architecture, the bit-width of a shared node is determined by the largest bit-width of all of the nodes that share a node. As shown in Figure 8 the largest bit-widths of input and output of the B_0 nodes in all of the FOSS architectures are both 16 bits, so the input and output bit-widths of the B_0 node in the unified architecture are both 16 bits.

3.4. The Design of a Low Cost Multiplexer Design for the Mux-Based Routing Network. To share a node of the unified architecture for the operations of multiple transforms, additional multiplexers are required to route a correspondent input path for individual transform mode. If N transforms share a node, each input of a node has a maximum of N different input paths and two additional N-to-1 multiplexers are deployed in front of each node. If an input path is B-bit wide, an input requires a B-bit, N-to-1 multiplexer. As shown in Figure 9, one input of the shared node is 8-bit wide, so an 8-bit 6-to-1 multiplexer is required to route one of the 6 input paths to the input of the shared node. If a 1-bit 2-to-1 multiplexer requires a hardware unit, as shown in Figure 10(a), a 1-bit 6-to-1 multiplexer requires 5 hardware units as shown in Figure 10(b). Because an 8-bit 6-to-1 multiplexer requires eight 1-bit 6-to-1 multiplexers, an 8-bit 6-to-1 multiplexer requires 40 (8×5) hardware units.

In order to reduce the hardware cost and to alleviate routing congestion in a VLSI physical design for a mux-based routing network for the unified architecture, the number of input paths for a shared node must be minimized. The multiplexer in Figure 9 is redesigned as a low cost and low routing congestion multiplexer, as shown in Figure 11. In Figure 9, there are two input paths, X_{00} and X_{01}, for one input of a shared node, so an 8-bit 2-to-1 multiplexer is used to select input path X_{00} or X_{01}. In addition, a 1-bit 6-to-1 multiplexer is used to select 0 or 1 for the selected line of the 8-bit 2-to-1 multiplexer. Using an 8-bit 2-to-1 multiplexer requires 8 hardware units and a 1-bit 6-to-1 multiplexer

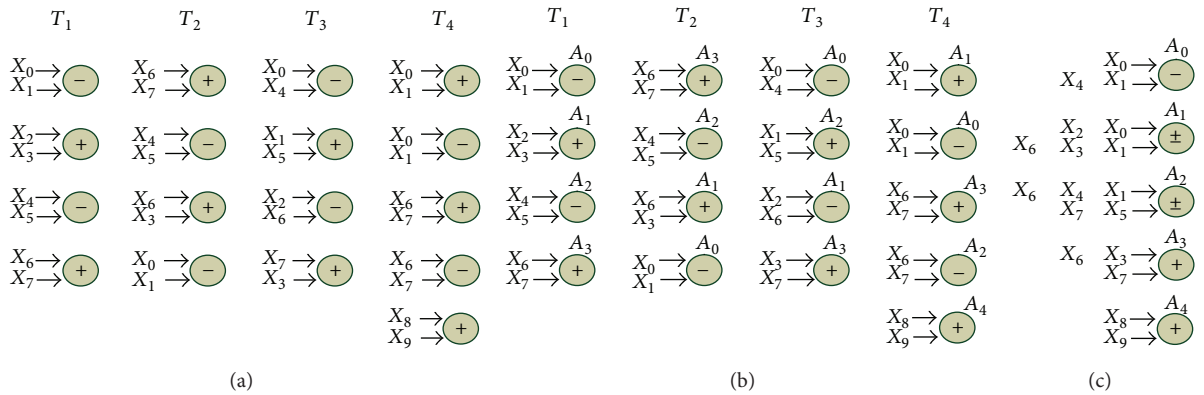

FIGURE 6: Sharing of a node in multiple FOSS architectures: (a) nodes of the same stage of different FOSS architectures, (b) determining which nodes of the same stage to share a node in a unified architecture, and (c) shared nodes of the same stage in a unified architecture.

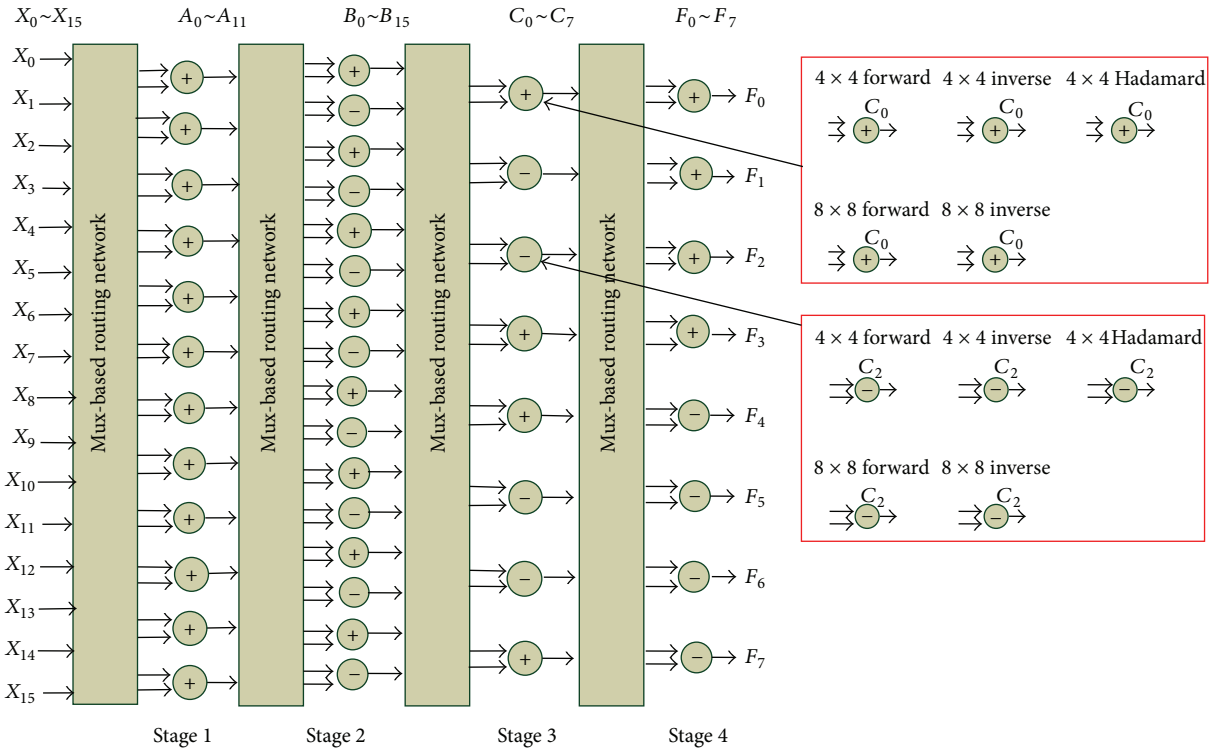

FIGURE 7: The operation decision for a node in the unified architecture.

TABLE 1: The input and output bit-widths for each transform of H.264.

	4 × 4 forward transform		4 × 4 inverse transform		8 × 8 forward transform		8 × 8 inverse transform	
	In	Out	In	Out	In	Out	In	Out
Row transform	9	12	16	16	9	12, 13	16	16
Column transform	12	15	16	16	13	16	16	16
	4 × 4 forward Hadamard		4 × 4 inverse Hadamard		2 × 2 forward Hadamard		2 × 2 inverse Hadamard	
	In	Out	In	Out	In	Out	In	Out
Row transform	16	16	16	16	16	16	16	16
Column transform	16	16	16	16	16	16	16	16

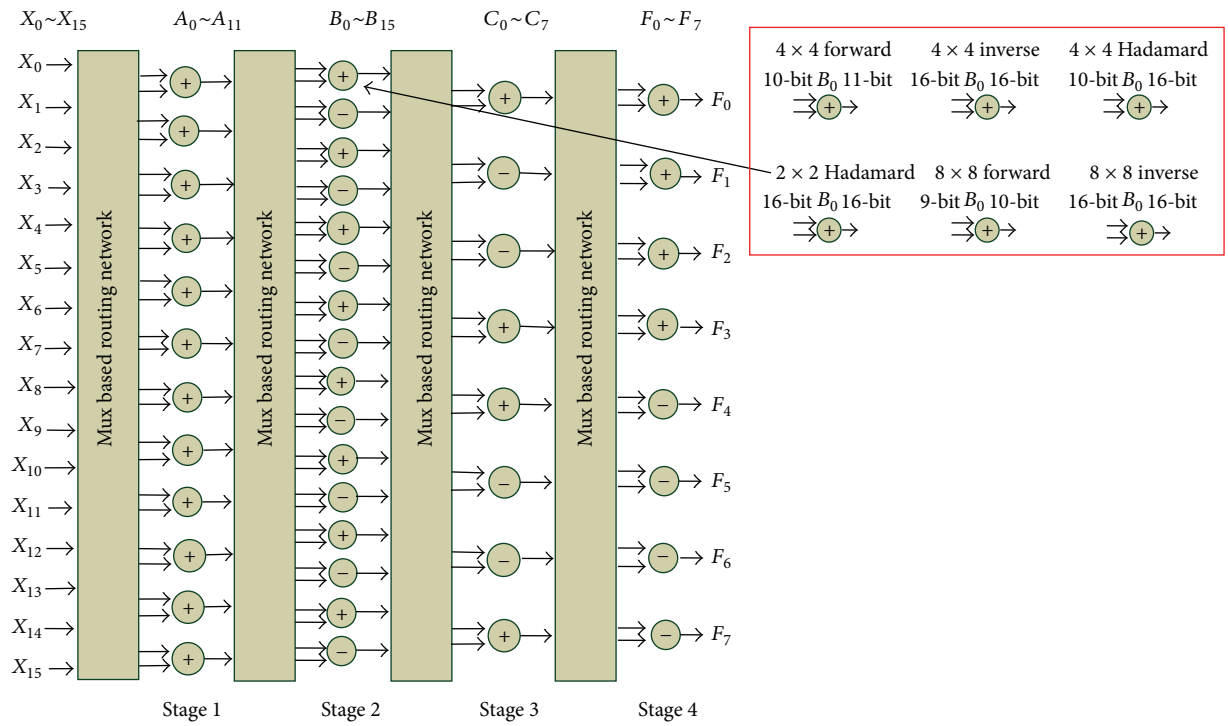

Figure 8: The bit-width decision for a node in the unified architecture.

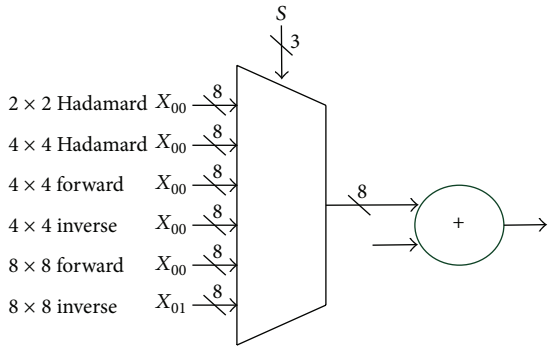

Figure 9: An 8-bit 6-to-1 multiplexer.

requires 5 hardware units, giving a total of 13 (8 + 5) hardware units. The cost of the multiplexer in Figure 11 is 67.5% that of the multiplexer in Figure 9.

The multiplexers required for a shared node result not only in extra hardware cost, but also in routing congestion for the unified architecture. The number of input paths for a 1-bit 6-to-1 multiplexer is 11, as shown in Figure 10(b), and that for an 8-bit 6-to-1 multiplexer is 88, as shown in Figure 9, which could cause clustering of the routing wires in particular areas. The routing congestion can be mitigated using the proposed multiplexer design, wherein the number of input paths is only 33, as shown in Figure 11. In addition, a good floor plan that uniformly distributes the multiplexers around the chip can disperse the routing wires. Note that additional latency is incurred by the multiplexers of the routing network that serialize the operations for different transform modes.

3.5. The Architecture-Unification Technique for All the H.264 Transform FOSS Architectures. The architecture-unification technique consists of count decisions for the unified architecture stages and the shared nodes, node sharing for the unified architecture for all FOSS architectures, the operation and the bit-width decisions for a shared node, and the design of a low cost multiplexer for the mux-based routing network. When this process is used for all of the H.264 transform architectures designed using the FOSS technique, Figure 12 shows the unified architecture for 2D 2 × 2 Hadamard transform, 2D 4 × 4 forward transform, 2D 4 × 4 inverse transform, 1D 8 × 8 forward transform, and 1D 8 × 8 inverse transform FOSS architectures.

4. Complexity and Performance Analysis

4.1. The Computational Complexity of the Original Transforms, the FOSS Architectures, and the Unified Architecture. The computational complexities of the original transforms, the FOSS architectures, and the unified architecture are analyzed. The computational complexities of the original transforms and the FOSS architectures are compared in Tables 2 and 3. As shown in Table 2, a total of 844 adders and 912 multipliers are needed for all of the six original H.264 transforms. However, only 24 sub/adders, 96 adders, 84 subtractors, and 60 shifters are needed for all of the six FOSS architectures that reduce the cost by sharing the same operations and by replacing the multipliers by adders and shifters, as shown in Table 3. The computational complexities of the FOSS architectures and the unified architecture are compared in Tables 3 and 4. Table 4 shows the computational complexity of the unified

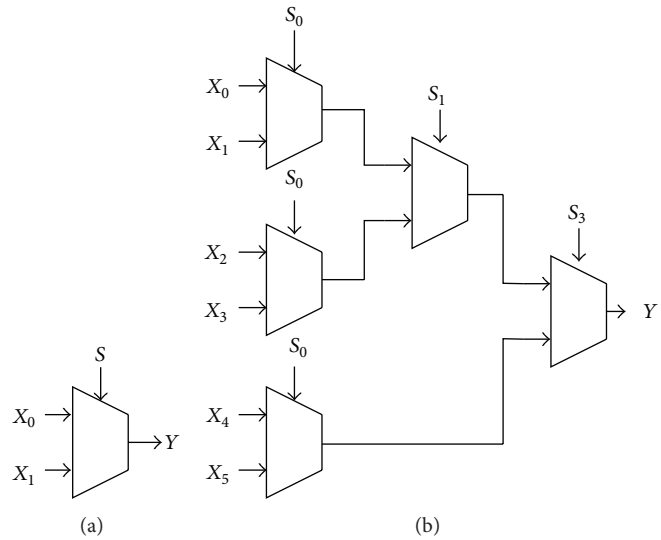

Figure 10: (a) A 1-bit 2-to-1 multiplexer. (b) A 1-bit 6-to-1 multiplexer.

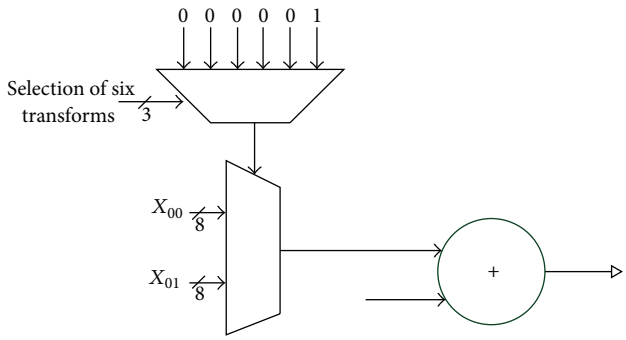

Figure 11: A low cost multiplexer design for the mux-based routing network.

Table 2: The computational complexity of the original transforms for H.264.

	Adder	Multiplier
1D 8 × 8 forward	56	64
1D 8 × 8 inverse	56	64
2D 4 × 4 forward	240	256
2D 4 × 4 inverse	240	256
2D 4 × 4 Hadamard	240	256
2D 2 × 2 Hadamard	12	16
Total	844	912

architecture, which only requires 28 adders, 16 subtractors, and 40 shifters and needs no sub/adders.

4.2. The Hardware Cost and Performance of the FOSS Architectures and the Unified Architecture. In order to implement the unified architecture, a front-end cell-based design flow is employed for logic design, simulation, and verification of VLSI implementation. The proposed FOSS architectures and the unified architecture are firstly realized using Verilog RTL code, and then a ModelSim EDA tool is used for the RTL functional simulation. Synopsys Design Compiler is used for logic synthesis and the standard cell library used is the UMC 0.18 μm Artisan cell library. The hardware cost and the performance of the FOSS architectures and the unified architecture are analyzed. The gate count is calculated using

$$\text{Gate Count} = \text{Synthesized area}/\text{NAND gate area}. \quad (12)$$

Because transform matrix multiplication uses different hardware architectures, the gate counts for the original transforms are not known. Table 5 shows the gate counts for the FOSS architectures and Table 6 shows those for the unified architecture, which has the gate count that is 36% less than the total gate count of all of the six FOSS architectures. Because the six FOSS architectures share the unified architecture, using the proposed architecture-unification technique, the number of gates saved for an individual transform is not known, but the total number of gates saved is the only way

Table 3: The computational complexity of the FOSS architectures for H.264 transforms.

	Sub/adder	Adder	Subtractor	Shifter
1D 8 × 8 forward	0	20	16	10
1D 8 × 8 inverse	0	24	16	18
2D 4 × 4 forward	8	16	16	16
2D 4 × 4 inverse	8	16	16	16
2D 4 × 4 Hadamard	8	16	16	0
2D 2 × 2 Hadamard	0	4	4	0
Total	24	96	84	60

Table 4: The computational complexity of the unified architecture for H.264 transforms.

	Adder	Subtractor	Shifter
Unified architecture	28	16	40

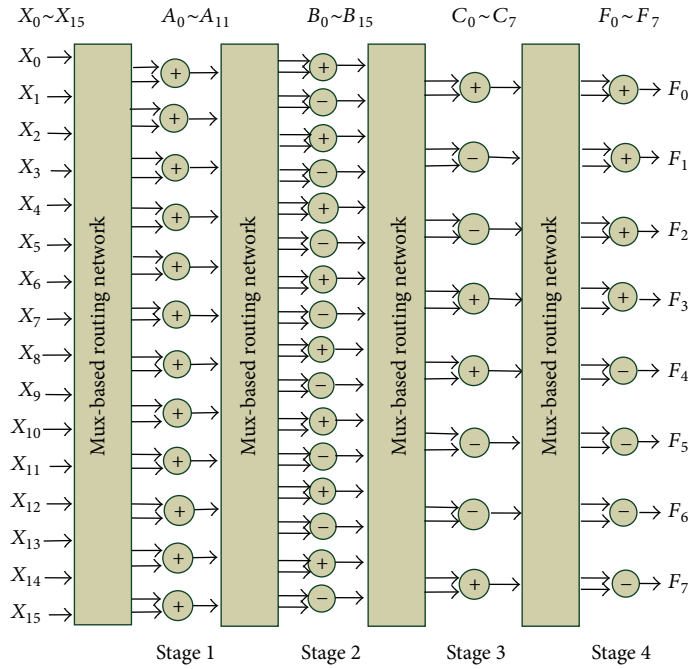

Figure 12: The unified architecture for all H.264 transforms.

Table 5: A comparison of the gate count, frequency, latency, and throughput for H.264 transform FOSS architectures.

	Gate count	Freq. (MHz)	Latency	Throughput (pixels/cycle)
1D 8 × 8 forward	2853	313	1	8
1D 8 × 8 inverse	5096	292	1	8
2D 4 × 4 forward	4299	306	1	8
2D 4 × 4 inverse	5821	285	1	8
2D 4 × 4 Hadamard	5437	302	1	8
2D 2 × 2 Hadamard	1032	385	1	4
Total	25438	N/A	N/A	N/A

Table 6: The gate count, frequency, latency, and throughput for the unified architecture.

	Gate count	Freq. (MHz)	Latency	Throughput (pixels/cycle)
Unified architecture	16308	275	1	Listed in Table 7

to measure the total reduction in hardware cost that results from the unification technique.

Since there is no register in either the FOSS architectures or the unified architecture, the critical path is the longest path of all of the paths from the input to the output of the architecture. The frequency is the reciprocal of the critical path. Because of the delay due to the mux-based routing network, the critical path for the unified architecture is slightly longer than that for any FOSS architecture. In other words, the frequency of the unified architecture is lower than that of any FOSS architecture. The frequency range of the FOSS architectures is from 285 MHz to 385 MHz, while the unified architecture still processes up to 275 MHz, as shown in Tables 5 and 6. Since there is no register in either the FOSS architectures or the unified architecture, the latencies are all one clock cycle. The throughput for the 2D 2 × 2 Hadamard transform FOSS architecture is 4 pixels/cycle, but those of the other transform FOSS architectures are all 8 pixels/cycle.

Table 7: The throughput for the unified architecture for H.264 transforms.

	Freq.	Throughput (pixels/cycle)
1D 8 × 8 forward	275 MHz	8
1D 8 × 8 inverse	275 MHz	8
2D 4 × 4 forward	275 MHz	8
2D 4 × 4 inverse	275 MHz	8
2D 4 × 4 Hadamard	275 MHz	8
2D 2 × 2 Hadamard	275 MHz	4

5. Conclusion

In this paper, a systematic hardware sharing method that allows a unified architecture for H.264 transforms is presented. A FOSS architecture design technique is presented to reduce the hardware cost for each H.264 transform. The basic idea is to systematically synthesize architecture, to share all of the same operations in a matrix multiplication, and to allow a reduction in hardware cost. In total 844 adders and 912 multipliers are required for all of the six H.264 transform matrix multiplications. However, only 24 sub/adders, 96 adders, 84 subtractors, and 60 shifters are

required for all of the six FOSS architectures, which reduces the cost by sharing the same operations and by replacing all of the multipliers by adders and shifters. When all of the six FOSS architectures of the H.264 transforms are determined, an architecture-unification design flow is then proposed that unifies all of the low cost transform FOSS architectures into a single architecture to eliminate the redundant hardware. The unified architecture only requires 28 adders, 16 subtractors, 40 shifters, and the proposed mux-based routing network. The gate count for the unified architecture is 16308, which is 36% less than the total gate count for all of the six FOSS architectures. The frequency range of the FOSS architectures is from 285 MHz to 385 MHz, while the unified architecture still processes up to 275 MHz. Since there is no register in either the FOSS architectures or the unified architecture, the latencies are all one clock cycle. Throughput for the 2D 2×2 Hadamard transform is 4 pixels/cycle, but those of the other transforms are all 8 pixels/cycle. In addition, the proposed hardware sharing method can also be used to construct a unified architecture for multitransform coding of other international video coding standards, such as VC-1, MPEG-1/2/4, and even the next generation high efficiency video coding (HEVC) that allows a reduction in hardware cost.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] "Coding of still pictures," ISO/IEC JTC 1/SC 29/WG 1, 2009.

[2] Video Coding Standard, "Information technology—coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s—part 2: video," ISO/IEC 11172-2 MPEG-1, 1993.

[3] "Video coding standard, information technology—generic coding of moving pictures and associated audio information: video," ISO/IEC 13818-2 MPEG-2, 1995.

[4] Video Coding Standard, "Information tchnology—coding of audio-visual objects—part 2: visual," ISO/IEC 14496-2 MPEG-4, 2004.

[5] A. Puri, X. Chen, and A. Luthra, "Video coding using the H.264/MPEG-4 AVC compression standard," *Signal Processing: Image Communication*, vol. 19, no. 9, pp. 793–849, 2004.

[6] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-complexity transform and quantization in H.264/AVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 598–603, 2003.

[7] S. Gordon, D. Marple, and T. Wiegand, "Simplified use of 8×8 transforms—updated proposal and results," in *Proceedings of the 11th Meeting JVTK028*, Munich, Germany, March 2004.

[8] T. Wang, Y. Huang, H. Fang, and L. Chen, "Parallel 4×4 2D transform and inverse transform architecture for MPEG-4 AVC/H.264," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 800–803, May 2003.

[9] 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 *Proceedings of the IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS '04)*, pp. 1141–1144, December 2004.

[10] 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 *Proceedings of the IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test (VLSI-TSA-DAT '05)*, pp. 291–294, April 2005.

[11] H.-Y. Lin, Y.-C. Chao, C.-H. Chen, B.-D. Liu, and J.-F. Yang, "Combined 2-D transform and quantization architectures for H.264 video coders," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '05)*, pp. 1802–1805, Kobe, Japan, May 2005.

[12] K. H. Chen, J. I. Guo, and J. S. Wang, "A high-performance direct 2-D transform coding IP design for MPEG-4 AVC/H.264," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 4, pp. 472–483, 2006.

[13] C.-P. Fan, "Fast 2-dimensional 4×4 forward integer transform implementation for H.264/AVC," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 3, pp. 174–177, 2006.

[14] T. Dias, S. López, N. Roma, and L. Sousa, "Scalable unified transform architecture for advanced video coding embedded systems," *International Journal of Parallel Programming*, vol. 41, no. 2, pp. 236–260, 2013.

[15] J. A. Michell, J. M. Solana, and G. A. Ruiz, "A high-throughput ASIC processor for 8×8 transform coding in H.264/AVC," *Signal Processing: Image Communication*, vol. 26, no. 2, pp. 93–104, 2011.

[16] M. Sharma, H. D. Tiwari, and Y. B. Cho, "Low cost high throughput pipelined architecture of 2-D 8×8 integer transforms for H.264/AVC," *International Journal of Electronics*, vol. 100, no. 8, pp. 1033–1045, 2013.

[17] Y. K. Lai and Y. F. Lai, "A reconfigurable IDCT processor architecture for video coding applications," in *Proceedings of the International Conference on Manufacturing and Engineering Systems (MES '09)*, pp. 469–472, December 2009.

[18] C.-P. Fan and G.-A. Su, "Efficient low-cost sharing design of fast 1-D inverse integer transform algorithms for H.264/AVC and VC-1," *IEEE Signal Processing Letters*, vol. 15, pp. 926–929, 2008.

[19] Y.-C. Chao, H.-H. Tsai, Y.-H. Lin, J.-F. Yang, and B.-D. Liu, "A novel design for computation of all transforms in H.264/AVC decoders," in *Proceedings of the IEEE International Conference onMultimedia and Expo (ICME '07)*, pp. 1914–1917, July 2007.

[20] G.-A. Su and C.-P. Fan, "Low-cost hardware-sharing architecture of fast 1-D inverse transforms for H.264/AVC and AVS applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 55, no. 12, pp. 1249–1253, 2008.

[21] H. Qi, Q. Huang, and W. Gao, "A low-cost very large scale integration architecture for multistandard inverse transform," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 7, pp. 551–555, 2010.

[22] C.-P. Fan, C.-H. Fang, C.-W. Chang, and S.-J. Hsu, "Fast multiple inverse transforms with low-cost hardware sharing design for multistandard video decoding," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 8, pp. 517–521, 2011.

[23] M. Martuza and K. A. Wahid, "Low cost design of a hybrid architecture of integer inverse DCT for H. 264, VC-1, AVS, and HEVC," *VLSI Design*, vol. 2012, Article ID 242989, 10 pages, 2012.

[24] K. A. Wahid, M. Martuza, M. Das, and C. McCrosky, "Efficient hardware implementation of 8×8 integer cosine transforms for multiple video codecs," *Journal of Real-Time Image Processing*, vol. 8, no. 4, pp. 403–410, 2013.

[25] C.-Y. Huang, L.-F. Chen, and Y.-K. Lai, "A high-speed 2-D transform architecture with unique kernel for multi-standard video applications," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '08)*, pp. 21–24, May 2008.

[26] S. Shen, W. Shen, Y. Fan, and X. Zeng, "A unified forward/inverse transform architecture for multi-standard video codec design," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 96, no. 7, pp. 1534–1542, 2013.