

# National Chiao Tung University

Department of Electronics Engineering

Master Thesis



Design of A Low Power Inverse Integer Transform for H.264/AVC  
Decoding Applications

Student : Hüseyin Demirkaya

Advisor : Prof. Chen-Yi Lee

**July 2011**

# Design of A Low Power Inverse Integer Transform for H.264/AVC Decoding Applications

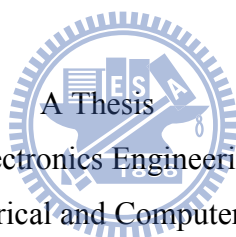
研 究 生：王英杰

Student : Hüseyin Demirkaya

指導教授：李鎮宜

Advisor : Prof. Chen-Yi Lee

國立交通大學  
電子工程學系 電子研究所 碩士班  
碩 士 論 文



Submitted to Department of Electronics Engineering & Institute of Electronics  
College of Electrical and Computer Engineering  
National Chiao Tung University  
in Partial Fulfillment of the Requirements  
for the Degree of Master  
in

Electronics Engineering

July 2011

Hsinchu, Taiwan, Republic of China

中華民國一〇一年七月

# Design of A Low Power Inverse Integer Transform for H.264/AVC Decoding Applications

Student : Hüseyin Demirkaya

Advisor : Prof. Chen-Yi Lee

Department of Electronics Engineering  
National Chiao Tung University

## ABSTRACT



In this thesis, we adopted various new fast butterfly algorithms and hardware architectures for low power Inverse Integer Transform (IIT) in H.264/AVC Main/High Profile video decoding. In our new fast algorithms we use matrix decomposition method to reduce the complexity of inverse integer transforms to reduce the power consumption, hardware cost and raise hardware efficiency in H.264/AVC Main/High Profile video decoding. Matrix decomposition utilizes the permutation matrices. The proposed design supports 4x4, 2x2/4x4 Hadamard and 8x8 inverse transforms.

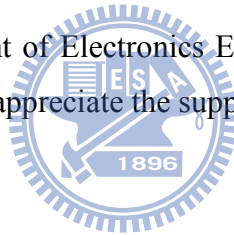
We integrate the same parts of the three transforms to reduce the power consumption and hardware area and the cost. Finally, we can use the proposed hardware design to handle the video coding with the 1080 HD @30fps and also QFHD @30fps video format. The proposed hardware architectures achieve power consumption and hardware cost for 4x4, Hadamard, 8x8 inverse integer transform and hardware sharing design at  $56.45\mu\text{W}$ ,  $46.85\mu\text{W}$ ,  $0.21\text{mW}$ ,  $0.31\text{mW}$ , and for the area 0.9k, 0.87k, 4.2k, 4.6k, respectively.

## *Acknowledgements*

First of all, I want to thank Prof. Chen-Yi Lee for very valuable comment and initiating an interesting research project and for providing excellent support.

Next, I would like to thank my fellow researchers in the Si2 laboratory. It has been a great pleasure working with them for the past two years. I would like to thank to my all Lab. People especially Yao Li for all their support and help. I wish them all great success in their future academic and professional lives.

I am also beholden to the Department of Electronics Engineering for providing support in the form of an assistantship. I also greatly appreciate the support of Himax Technologies Inc.

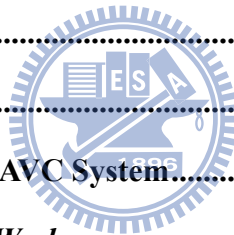


# Contents

---

<b>Chapter 1</b>	<b>Introduction</b> .....	<b>1</b>
1.1	Introduction of H.264/AVC decoding flow.....	3
1.2	Motivation & Design Challenges .....	5
1.3	Thesis Organization .....	6
<b>Chapter 2</b>	<b>Related Works</b> .....	<b>7</b>
2.1	Inverse Integer Transform Algorithm.....	7
2.1.1	Overview of the Inverse Integer Transforms.....	7
2.1.2	Traditional 4x4 Inverse Integer Transform .....	11
2.1.3	Traditional Inverse Hadamard Integer Transform .....	12
2.1.4	Traditional 8x8 Inverse Integer Transform.....	13
2.1.5	Traditional Hardware Sharing Design.....	16
2.2	H.264 Profiles and Levels .....	18
2.3	Previous Works.....	21
2.3.1	Parallel 4x4 transform and inverse transform Architecture for MPEG-4 AVC/H.264 [5].....	21
2.3.2	Low Cost Hardware sharing Architecture of Fast Inverse Transforms for H.264/AVC and AVS Applications [8].....	22
2.3.3	A High Performance Inverse integer Transform Architecture for the H.264/AVC Decoder[12].....	23
2.3.4	Configurable, Low-power Design for Inverse Integer Transform in H.264/AVC[16].....	24
2.3.5	A Reconfigurable IDCT Architecture for Universal Video Decoders[17].....	26
2.4	Summary .....	29
<b>Chapter 3</b>	<b>Proposed Algorithm &amp; Architecture</b> .....	<b>31</b>
3.1	Fast 4x4 Inverse Integer Transform .....	31

3.1.1	Fast 4x4 Inverse Integer Transform Algorithm .....	31
3.1.2	Fast 4x4 Inverse Integer Transform Architecture .....	34
3.2	Fast Inverse Hadamard Integer Transform .....	35
3.2.1	Inverse Hadamard Integer Transform Algorithm .....	35
3.2.2	Inverse Hadamard Integer Transform Hardware Architecture.....	36
3.3	Fast 8x8 Inverse Integer Transform .....	38
3.3.1	Fast 8x8 Inverse Integer Transform Algorithm .....	38
3.3.2	Fast 8x8 Inverse Integer Transform Hardware Architecture .....	42
3.4	Hardware Sharing Algorithm & Architecture .....	45
3.4.1	Comparison And Implementation Of Hardware Sharing Architecture .....	46
3.5	Summary and Comparison With Related Works .....	48
<i>Chapter 4</i>	<i>System Integration</i> .....	52
4.1	System Specification.....	52
4.2	The Integration with H.264/AVC System.....	53
<i>Chapter 5</i>	<i>Conclusion and Future Works</i> .....	56
5.1	Conclusion.....	56
5.2	Future Work .....	57
<b>References</b>	<b>58</b>	

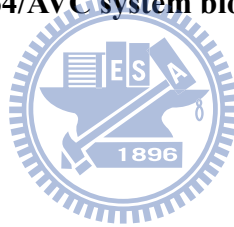


# Figure Index

---

Figure 1. System architecture of H.264/AVC decoder .....	1
Figure 2. Bit-stream structure of H.264/AVC.....	2
Figure 3. Block diagram of H.264/AVC decoder.....	3
Figure 4. Residuals (prediction errors) between current and reconstructed frame .....	7
Figure 5. Scanning order of residual blocks within a macroblock.....	9
Figure 6. H.264/AVC Encoding/Decoding flow diagram.....	9
Figure 7. Traditional 4x4 inverse integer transform.....	12
Figure 8. Traditional 4x4 inverse Hadamard integer transform.....	13
Figure 9. Traditional 8x8 inverse integer transform.....	15
Figure 10. Traditional hardware sharing design [19].....	16
Figure 11. High profile classification and features.....	18
Figure 12. H.264 decoder profiling results.....	19
Figure 13. (Re-designed) parallel transform architecture.....	21
Figure 14. Block diagram of the proposed hardware sharing architecture of fast 2x2, 4x4 and 8x8 inverse transforms for H.264/AVC and AVS with four pipeline phases [8] .....	22
Figure 15. 4x4 inverse transform hardware architecture [12].....	23
Figure 16. Functional block diagram: Configurable inverse integer transform unit. [16] ..	24
Figure 17. Data flow diagram for (a) M1, (b) M2, (c) M3, and (d) M4 cases. ....	25
Figure 18. Data flow diagram for (a) CM14, (b) CM24 and (c) CM34.....	26
Figure 19. Block diagram of reconfigurable inverse integer transform .....	27
Figure 20. Architecture of reconfigurable one dimensional inverse integer transform .....	27

<b>Figure 21. Architecture of adder kernel a) Even part and b) odd part.....</b>	<b>27</b>
<b>Figure 22. Implementation strategies of previous works .....</b>	<b>30</b>
<b>Figure 23. Pipeline hardware architecture for fast 4x4 inverse integer transform. ....</b>	<b>34</b>
<b>Figure 24. New fast algorithm of 4x4 inverse integer transform.....</b>	<b>35</b>
<b>Figure 25. Pipeline hardware architecture for fast Hadamard inverse integer transform..</b>	<b>36</b>
<b>Figure 26. New fast algorithm of 4x4 Hadamard inverse integer transform. ....</b>	<b>37</b>
<b>Figure 27. Pipeline hardware architecture for fast 8x8 inverse integer transform. ....</b>	<b>43</b>
<b>Figure 28. New fast algorithm of 8x8 inverse integer transform.....</b>	<b>43</b>
<b>Figure 29. Hardware sharing architecture for fast 8x8, 4x4 and Hadamard inverse integer transform. ....</b>	<b>46</b>
<b>Figure 30. The Integration with H.264/AVC system block diagram .....</b>	<b>53</b>





# Table Index

---

<b>Table 1. Motivation comparison of standards .....</b>	<b>5</b>
<b>Table 2.Coding tools in different profiles of H.264/AVC standard.....</b>	<b>20</b>
<b>Table 3. Supporting features comparison .....</b>	<b>29</b>
<b>Table 4. Architecture Comparisons for Fast Inverse Integer Transforms.....</b>	<b>47</b>
<b>Table 5. Normalization of Power consumption to UMC 0.18um and TSMC 0.18um .....</b>	<b>49</b>
<b>Table 6. Synthesis results and Comparison .....</b>	<b>50</b>
<b>Table 7.The specification of Video decoder .....</b>	<b>52</b>
<b>Table 8. Time required to decoding full HD and HDTV frame with different MB. Frame with YUV420 is used.....</b>	<b>55</b>



# Chapter 1

## Introduction

---

H.264/AVC is the state-of-the-art video compression standard of the ITU-T Video Coding Experts Group and ISO/IEC Moving Picture Experts Group (MPEG) in current video applications and is often exploited in electronics devices to achieve better video compression performance. The objective of the H.264/AVC is to deliver high quality video at lower bit rates than the previous standards. One of the tools being adopted is inverse integer discrete cosine transform. The video compression efficiency achieved in H.264/AVC standard is not a result of any single feature but a combination of a number of codec tools. As it is shown in system architecture block diagram of an H.264/AVC decoder in Figure 1, the inverse integer transform algorithm [2] is one of the coding tools.

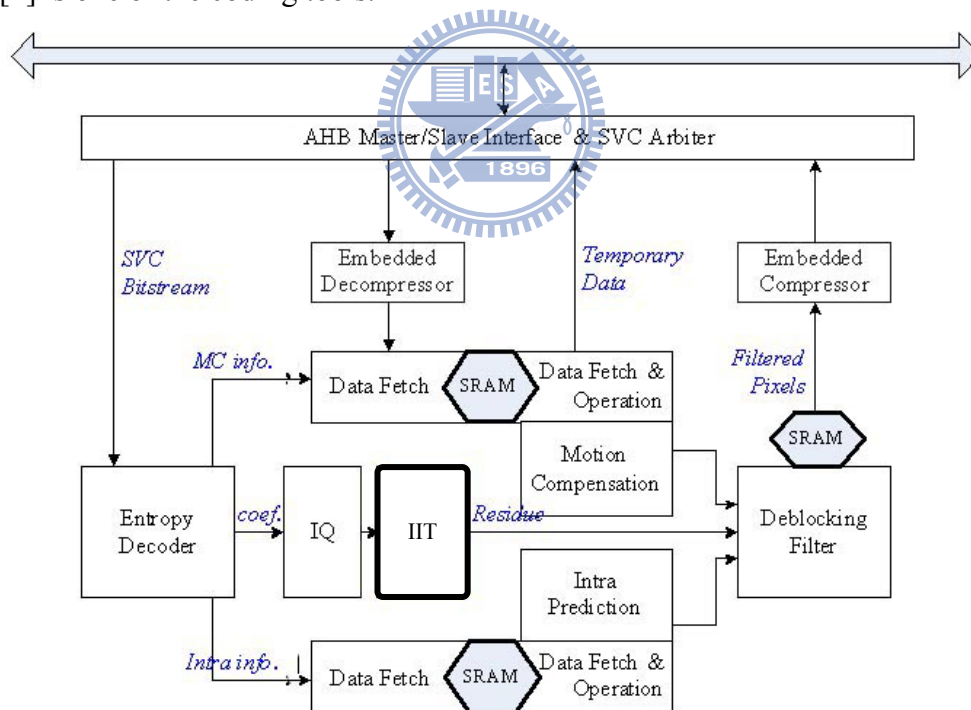


Figure 1. System architecture of H.264/AVC decoder

To quickly compress video data in spatial domain, H.264/AVC employs 4x4 integer transforms which use only integer arithmetic with signed additions and shifts to replace the costly multiplication. Small block-size transform tends to reduce the computational complexity

and ringing artifacts. However, for high-quality video, large block-size transform must be used not only to preserve fine details of the image but also to obtain the better energy compaction. High-definition (HD) applications adopt main profile, extended profile and high profile in H.264/AVC, and they require complicated design. Also, H.264/AVC offers various scalabilities to be adapted to the receipt condition of the data and the various multimedia applications [3]. Transform process of H.264 requires 8x8 transforms for high-definition applications, and 4x4 transforms for general applications. To meet scalabilities, transform module must process both 8x8 integer transform operations and 4x4 integer transform operations. Therefore, design of 8x8 transforms and 4x4 transforms into unified block is an important issue in H.264/AVC coder. High profile in H.264/AVC Fidelity Range Extension (FRExt), which is a new amendment added in H.264 standard, includes 8x8 integer transform and allows the decoder to adaptively choose between 4x4 and 8x8 transform for luma samples on MB level [4]. Based on the symmetric property of the integer transform matrix and matrix operations, which denote the row-column permutations and the matrix decompositions, the efficient fast 4x4 and 8x8 inverse transform developed [6]-[8]. For early-stage H.264/AVC such as the baseline or main profile, researchers mainly focused on developing the fast algorithm of 4x4 and 8x8 transforms and its implementation to improve performance with minimal area overhead in [10]-[13].

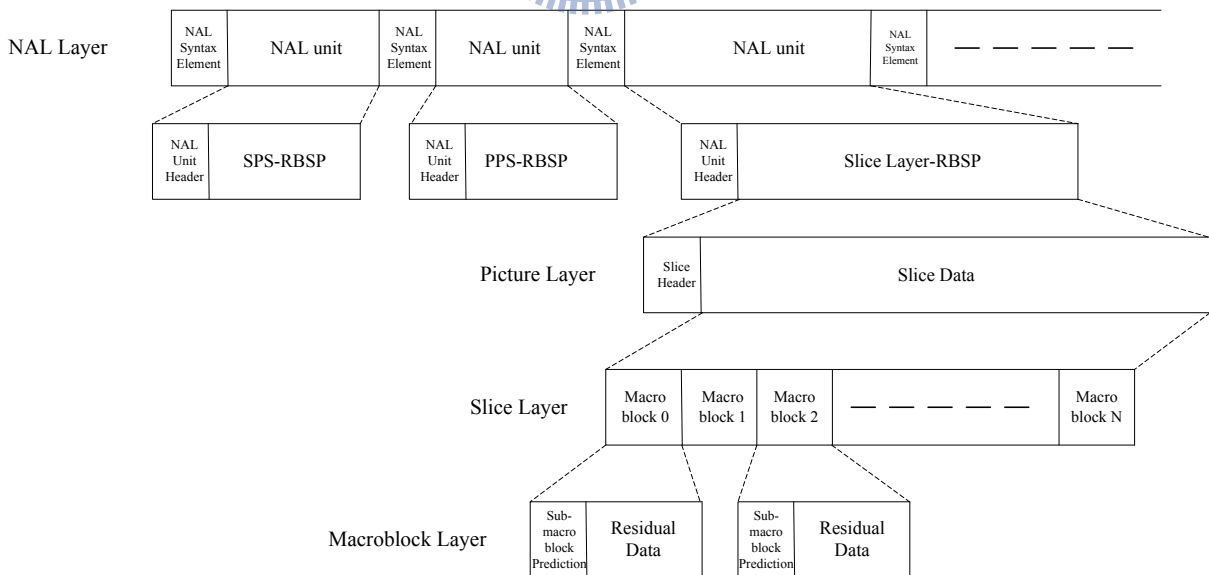


Figure 2. Bit-stream structure of H.264/AVC

In normal system architecture, the block of syntax parser employs in decoding the bit-stream on NAL layer, picture layer, and slice layer, given as Figure 2. Syntax element parser is also the top module to control all sub-system such as inverse integer transform, CABAD, VLD, intra-prediction, inter-prediction, and so on.

## 1.1 Introduction of H.264/AVC decoding flow

In common with earlier coding standards, H.264 does not explicitly define a CODEC (encoder/decoder pair) but rather defines the syntax of an encoded video bitstream together with the method of decoding this bitstream. In practice, a compliant decoders are likely to include the functional elements shown in Figure 3. With the exception of the deblocking filter, most of the basic functional elements (prediction, inverse transform and inverse quantization) are present in previous standards (MPEG-1, MPEG-2, MPEG-4, H.261, H.263) but the important changes in H.264 occur in the details of each functional block.

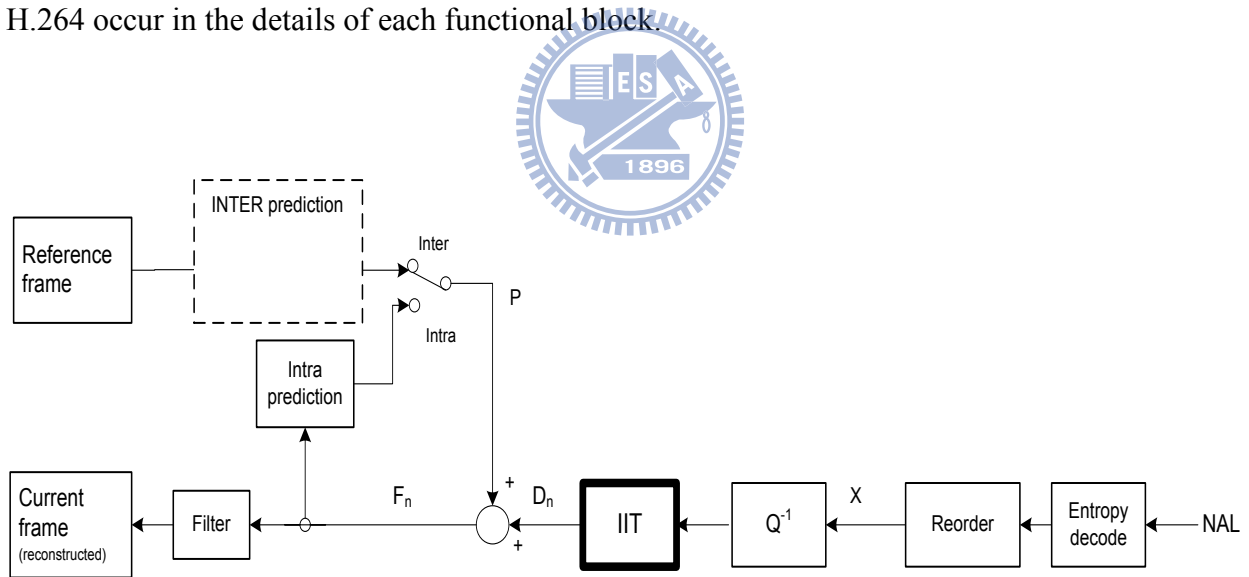
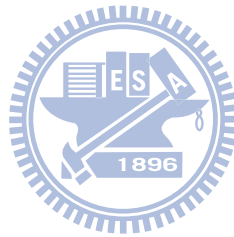


Figure 3. Block diagram of H.264/AVC decoder

The decoder receives a compressed bitstream from the NAL and entropy decodes the data elements to produce a set of quantized coefficients  $X$ . These are scaled and inverse transformed to give  $D_n$ . Using the header information decoded from the bitstream, the decoder creates a

prediction block (PRED), identical to the original prediction formed in the encoder. PRED is added to  $D_n$  to produce  $F_n$  which is filtered to create each decoded block (Current frame).



## 1.2 Motivation & Design Challenges

The transform process that converts image or motion compensated residual data into another domain. H.264 supports several inverse integer transforms. Table 1 shows the motivation comparison with the previous standard. Our target is to reduce the complexity of inverse integer transform and make it fast. If we can reduce transform complexity, we can also reduce power consumption and hardware cost, moreover, increase the throughput. By applying the concept of hardware sharing, power-area requirement for the hardware implementation of the inverse integer transform will be reduced by sharing the hardware resources. Optimizing 4x4, 8x8 and Hadamard inverse transforms algorithm of the H.264/AVC decoder, we obtain low power consumption, high performance and small area design. To reduce power consumption and enhance performance of the transform with a minimum area overhead remain the design challenges in H.264/AVC video standard.

Table 1. Motivation comparison of standards

Feature/Standard	MPEG-1	MPEG-2	MPEG-4 part 2	H.264/MPEG part 10
Macro block size	16x16	16x16 (frame mode) 16x8 (field mode)	16x16	16x16
Block size	8x8	8x8	16x16, 16x8, 8x8	16x16, 8x16, 16x8, 8x8,4x8, 8x4, 4x4
Inverse transform	8x8 DCT	8x8 DCT	8x8 DCT/Wavelet	4x4, 8x8, Hadamard Integer transform

## 1.3 Thesis Organization

An introduction of H.264/AVC video coding standards are given in this section. The rest of this thesis is organized as follows. Chapter 2 describes the related works for inverse integer transforms, the overview of H.264/AVC profiles and previous works. In Chapter 3, we describe our proposed algorithm and hardware architecture for 4x4, Hadamard and 8x8 inverse integer transform. In this chapter, we also implement the hardware sharing algorithm & architecture, and take an in-depth discussion about Comparison and implementation of hardware sharing architecture. Moreover, we show 4 kinds of Inverse integer transform module design including hardware sharing and according to our proposed algorithm, system integration architecture and comparison of the proposed design with others shows in 0. Finally, we make the conclusion and future works in the last Chapter 5.



# Chapter 2

## Related Works

---

In this chapter, we will describe the overview of the H.264/AVC traditional inverse integer transform algorithm for 4x4, Hadamard and 8x8 MB.

## 2.1 Inverse Integer Transform Algorithm

### 2.1.1 Overview of the Inverse Integer Transforms

H.264/AVC uses a macroblock (MB) as a basic data unit. Our input includes coefficients and flags decoded by the entropy decoder (CAVLC or CABAC). It contains luma part and chroma part. The inter-prediction or intra prediction module finds a macroblock which is similar to current one from reference or present frames. However, the founded MB usually does not perfectly match with the current one, and the differences are called residuals (or prediction error) as shown in Figure 4. The residuals are inversely transformed which are then reordered and entropy encoded. At the decoder side, these entropy-encoded coefficients are decoded back to coefficients. After reordering, coefficients are inversely transformed to residuals data. Finally the residuals are combined with prediction data to reconstruct a MB.

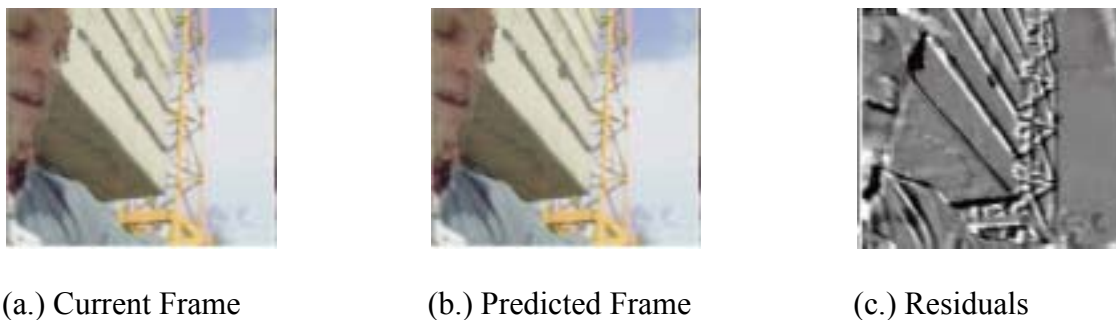
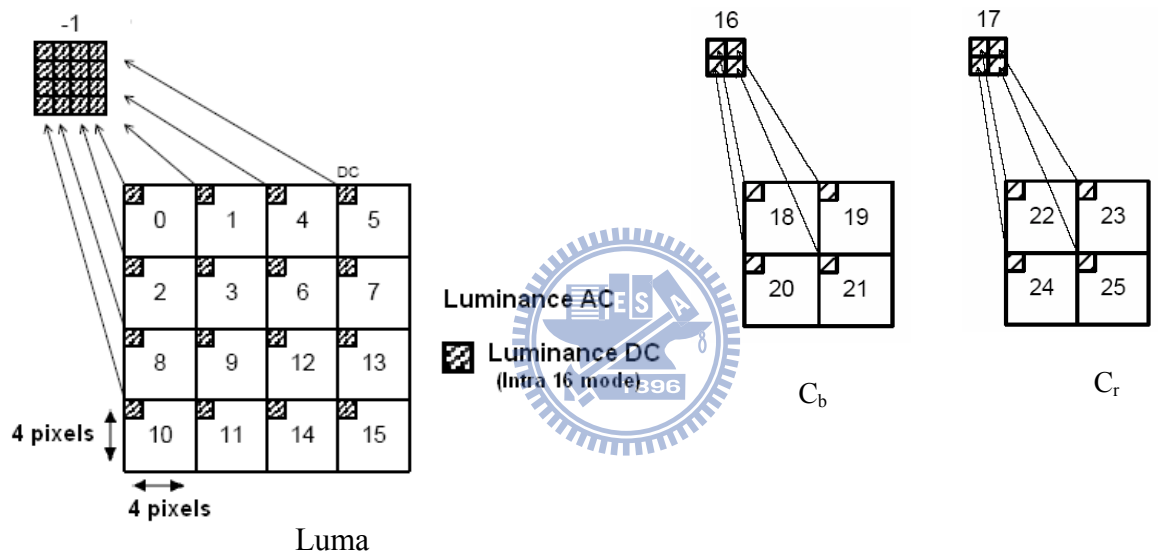


Figure 4. Residuals (prediction errors) between current and reconstructed frame

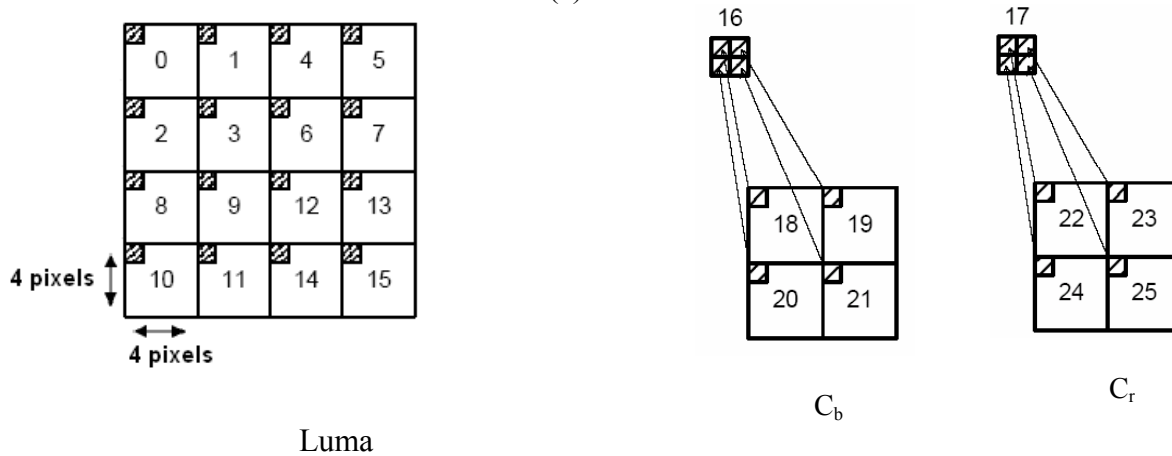
There are one 16x16 luma block and two 8x8 chroma blocks (Cb, Cr) within a macroblock. A 16x16 luma block can be divided into four 8x8 blocks, and each consists of four 4x4 blocks. A chroma 8x8 block contains four 4x4 blocks. In Figure 4, every 4x4 (or 2x2) block is numbered



according to decoding order. If a macroblock is coded by intra 16x16 prediction mode as shown in Figure 5(a), block -1 which contains DC coefficients of every 4x4 luma block will be processed first. The DC coefficients are filled back to upper-left corner of each 4x4 block in a 16x16 luma block. Next, the luma residual blocks 0-15 are processed. After luma block is decoded, chroma DC blocks 16 and 17 are processed, and filled back to upper-left corner of each 4x4 block in an 8x8 chroma block. Finally, chroma residual blocks 18-25 are processed. If current macroblock type is non-intra 16x16, the processing order is the same except that it has no luma DC block as shown in Figure 5(b).



(a) Intra 16x16 macroblock



(b) Non-Intra 16x16 macroblock

Figure 5. Scanning order of residual blocks within a macroblock

Three kinds of inverse integer transform are adopted depending on the type of residual blocks: 4x4 Hadamard transform for luma DC block (block -1), 2x2 Hadamard transform for chroma DC block (block16, 17), and 4x4 integer transform for all other types of 4x4 blocks (block 0-15, 18-25).Figure 6 shows the decoding flow diagram. We will emphasize on the decoder side.

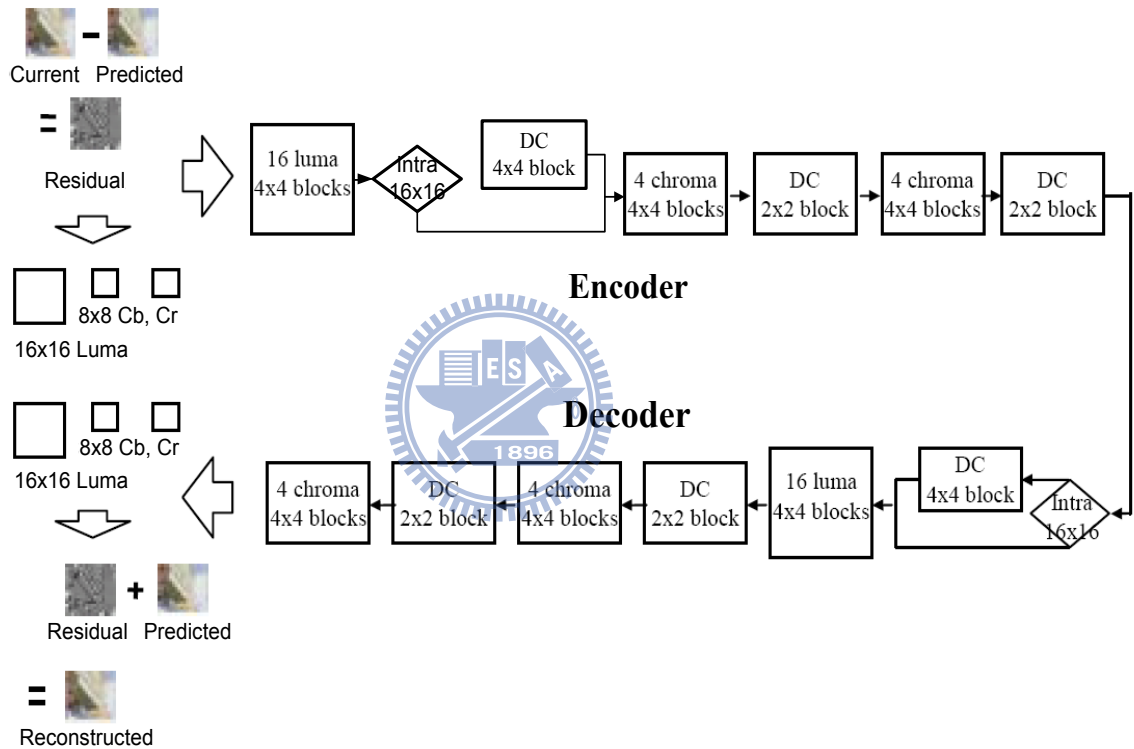
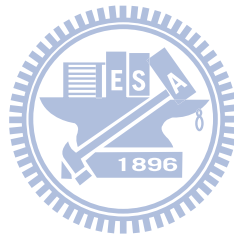


Figure 6. H.264/AVC Encoding/Decoding flow diagram

If 4x4 inverse transform is employed, the luma part is divided into one luma DC 4x4 block and 16 luma AC 4x4 blocks. On the other hand, if 8x8 transform is applied, the luma part is divided into four 8x8 blocks. The chroma part is divided into two chroma DC 2x2 blocks and eight chroma AC 4x4 blocks in both cases.

In the following sub-sections, we will describe the traditional 4x4, Hadamard and 8x8 inverse integer transform algorithms.



## 2.1.2 Traditional 4x4 Inverse Integer Transform

In the H.264/AVC standard, the inverse integer transform operates on 4x4 blocks of residual data after motion-compensated prediction or intra prediction [4]. However, only two types of 4x4 inverse transforms are defined for the H.264/AVC decoder. The first type is the 4x4 inverse integer transform, which is defined as Eq. 2.1, where the 4x4 inverse integer transform coefficient matrix  $A_{4i}$  defined as Eq. 2.2

$$X = \overline{A_{4i}}^T (Y \bullet E_i) \overline{A_{4i}} = A_{4i}^T F A_{4i} \quad \text{Eq. 2.1}$$

$$\overline{A_{4i}}^T = A_{4i} = \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \quad E_i = \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \quad \text{Eq. 2.2}$$

and  $F = Y \bullet E_i$

Then  $A_{4i}$  and  $E_i$  are given by and where  $a=1/2$  and  $b=\sqrt{2/5}$ . The “•” means each element of Y is multiplied by the scaling factor in the same position in matrix  $E_i$  [3]. Since the scaling matrix  $E_i$  could be merged into the inverse quantization and pre-scaled process to reduce the number of multiplication process. Figure 7 show the traditional 4x4 inverse integer transform algorithm.

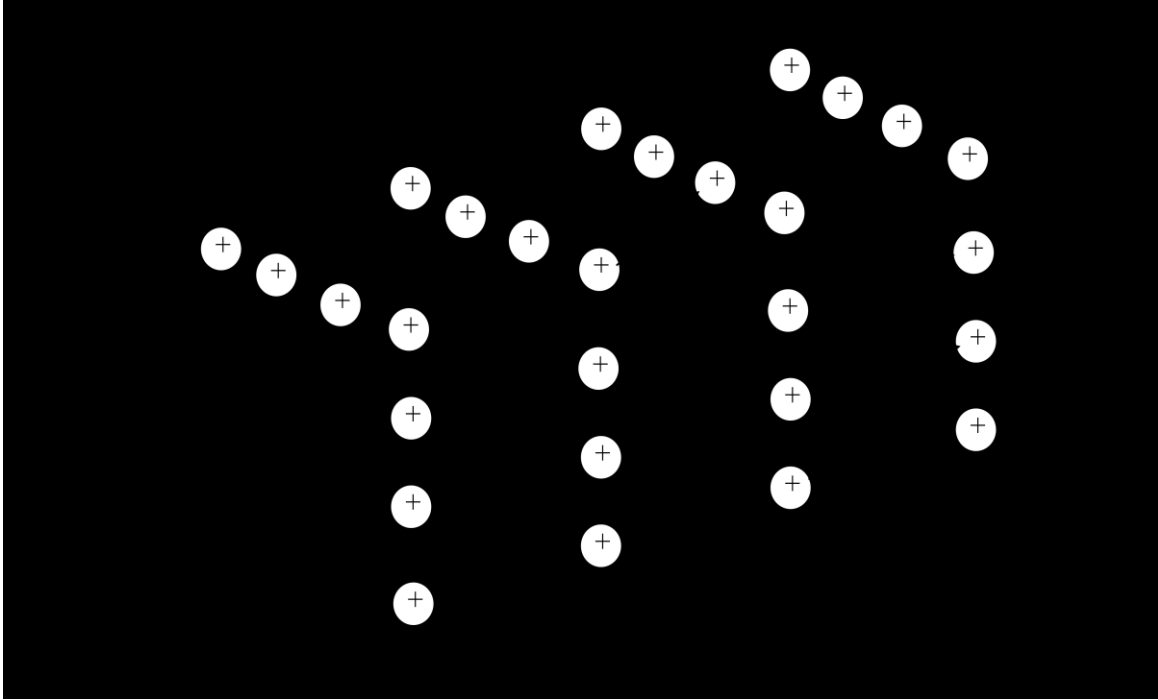


Figure 7. Traditional 4x4 inverse integer transform

### 2.1.3 Traditional Inverse Hadamard Integer Transform

The second type is the 4x4 inverse Hadamard transform (also known as the luma DC transform). The inverse Hadamard transform is defined as Eq. 2.3, where  $X_D$  is the 4x4 DC component of a 16x16 intra mode macroblock.

$$W_D = H_{4i} X_D H_{4i}^T \quad \text{Eq. 2.3}$$

The 4x4 inverse Hadamard integer transform coefficient matrix  $H_{4i}$  defined as Eq. 2.4 and Figure 8 shows the traditional two dimensional inverse Hadamard fast algorithm.

$$H_{4i} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \quad \text{Eq. 2.4}$$

The 2x2 Hadamard transform use the same formula for inverse integer transform as Eq. 2.5

$$\mathbf{X}_D = \mathbf{H}_{2i} \mathbf{W}_D \mathbf{H}_{2i}^T, \text{ with } \mathbf{H}_{2i} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \text{Eq. 2.5}$$

In the H.264/AVC standard, the 2x2 chroma DC transform is also defined. Since it is implied in the 4x4 inverse Hadamard transform.

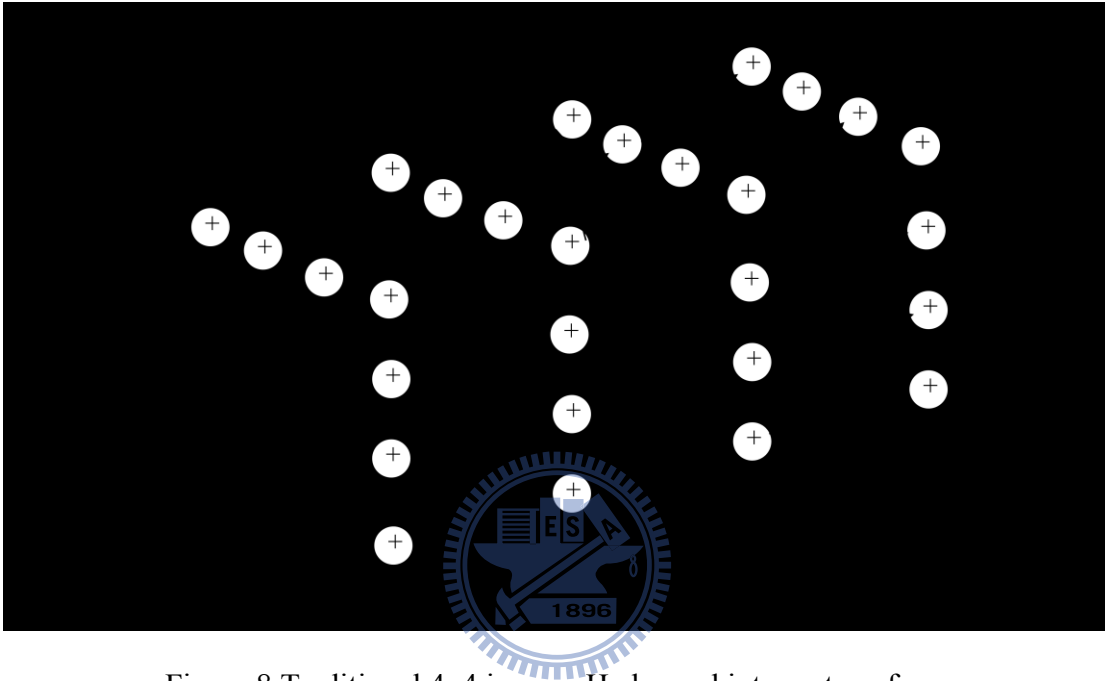


Figure 8. Traditional 4x4 inverse Hadamard integer transform

## 2.1.4 Traditional 8x8 Inverse Integer Transform

The 8x8 forward and inverse integer transforms can be performed in a similar with 4x4 manner. 8x8 forward integer transform can be realized by the following equivalent form as Eq. 2.6, where  $\tilde{E}_f$  is the scaling matrix. Meanwhile, 8x8 inverse integer transform is described as Eq. 2.7, where  $\tilde{E}_i$  is the scaling matrix.

$$\mathbf{Y} = \left( \mathbf{C}_{8f} \mathbf{X} \mathbf{C}_{8f}^T \right) \bullet \tilde{E}_f \quad \text{Eq. 2.6}$$

$$X = C_{8i} \left( Y \bullet \tilde{E}_i \right) C_{8i}^T = C_{8i} \bar{Y} C_{8i}^T \quad \text{Eq. 2.7}$$

The  $C_{8i}$  is the corresponding 8x8 inverse transform matrix and we note that  $C_{8i} = C_{8f}^T$ . Coefficient of 8x8 inverse integer transform for high profile is shown in Eq. 2.8. The 8x8 transforms are only applied to luma blocks.

$$C_{8i} = \begin{bmatrix} 8 & 12 & 8 & 10 & 8 & 6 & 4 & 3 \\ 8 & 10 & 4 & -3 & -8 & -12 & -8 & -6 \\ 8 & 6 & -4 & -12 & -8 & 3 & 8 & 10 \\ 8 & 3 & -8 & -6 & 8 & 10 & -4 & -12 \\ 8 & -3 & -8 & 6 & 8 & -10 & -4 & 12 \\ 8 & -6 & -4 & 12 & -8 & -3 & 8 & -10 \\ 8 & -10 & 4 & 3 & -8 & 12 & -8 & 6 \\ 8 & -12 & 8 & -10 & 8 & -6 & 4 & -3 \end{bmatrix} \quad \text{Eq. 2.8}$$

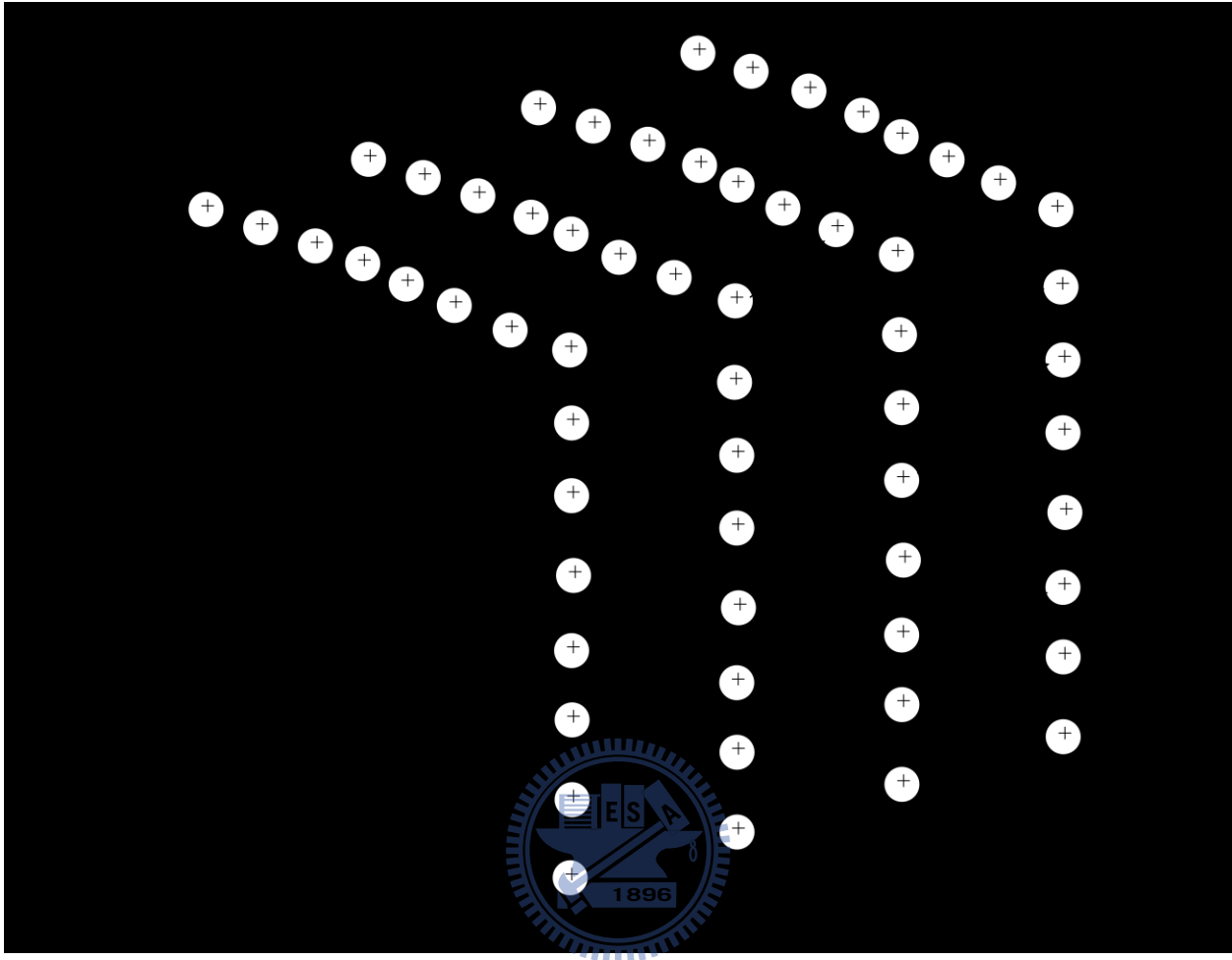


Figure 9. Traditional 8x8 inverse integer transform

In the previous section, we already know the H.264 integer inverse transform (4x4, Hadamard, 8x8) their principle and algorithms. For the implementation, the first one dimensional inverse integer transform block executes the transformation of row pixels and the second one dimensional inverse integer transform block performs the transformations of column pixels. Such as, Figure 9 is that the traditional 8x8 inverse integer transform method for implementation of hardware algorithm.



## 2.1.5 Traditional Hardware Sharing Design

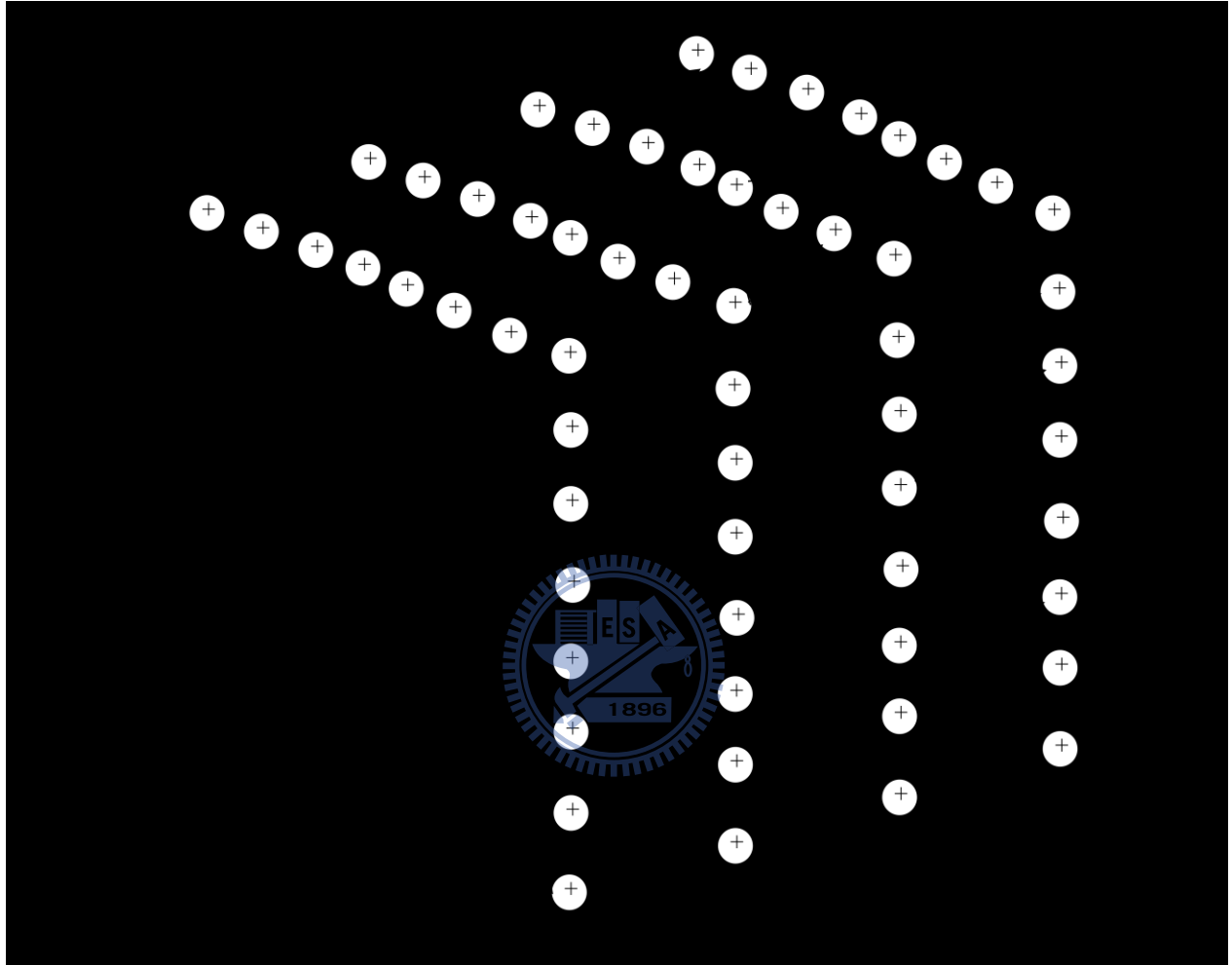


Figure 10. Traditional hardware sharing design [19]

In order to reduce the gate count required for the two different transform processors, using multi transform (hardware sharing) algorithm that combine the three transform units into one multiple function transform processor which can execute all the three transform operations in H.264. In traditional hardware sharing architecture shown in Figure 10, the one dimensional transform can be any type of the transform. By the observation of Figure 7 and Figure 8, we can find that every one-dimensional transform contains 8 arithmetical operations. In order to get a clear view of how to achieve hardware sharing transforms in a single design, we overlap Figure 7, Figure 8, and Figure 9 together. In Figure 10, all the adders have three inputs. It means that a

common input which is not changed by the transform type exists. Furthermore, Figure 10 is the fully extended of [19] into 64 pixels.



## 2.2 H.264 Profiles and Levels

H.264/AVC defines four profiles: baseline, extended, main and high profile. Baseline profile is usually used in low bit-rate applications. Extended profile, also called streaming profile, is designed for internet communication. Main profile is suitable for broadcast and storage applications. High profile, also called Fidelity Range Extension (FRExt), is intended for high resolution applications characterized by large block transform and large prediction blocks.

The high profile is further classified into four sub-profiles: High, High 10, High 4:2:2 and High 4:4:4, as depicted in Figure 11. These features include 8x8 luma transform, 8x8 spatial luma prediction, custom scaling matrix, deeper sample bits and lossless coding. Among them, the 8x8 luma transform is the key.

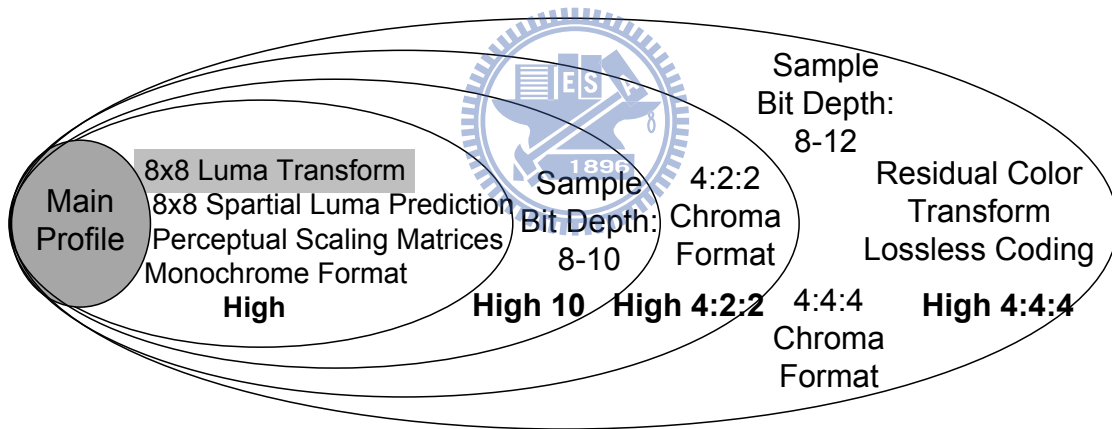


Figure 11. High profile classification and features

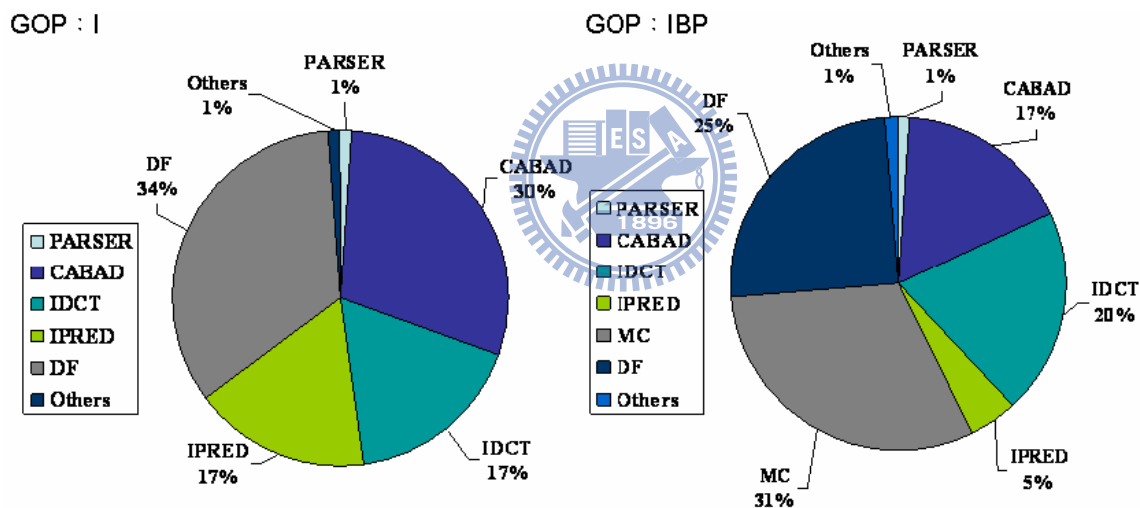
Figure 12 shows the profiling result of decoding a high profile video sequence. The inverse integer transform consumes about 17% to 20% of CPU time. Therefore, we design a low power inverse integer transform for integration into H.264/AVC decoder depicted in Figure 1.

Some important H.264 profiles and their special features are:

Baseline Profile: Only I and P type slices are present, only frame mode (progressive) picture types are present, Only CAVLC is supported.

Main Profile: Only I, P, and B type slices are present, Frame and field picture modes (in progressive and interlaced, modes) picture types are present, Both CAVLC and CABAC are supported, ASO is not supported, FMO is not supported.

High Profile: Only I, P, and B type slices are present, Frame and field picture modes (in progressive and interlaced modes) picture types are present, Both CAVLC and CABAC are supported, ASO is not supported, FMO is not supported, 8x8 transform supported, Scaling matrices supported.



- Reference software : JM11.0
- Reference frame : 2 frames
- QP : 28
- Resolution : 3840 x 2160
- Search range : 128
- Sequence : pedestrian
- Frame number : 60frames

Figure 12.H.264 decoder profiling results

All of these profiles also support mono chroma coded video sequences, in addition to typical 4:2:0 video. The difference in capability among these profiles is primarily in terms of supported sample bit depths and chroma formats. However, the high 4:4:4 profile additionally supports the

residual color transform and predictive lossless coding features are not found in any other profiles. The detailed capabilities of these profiles are show in Table 2.

Table 2.Coding tools in different profiles of H.264/AVC standard

<b>Coding Tools</b>	<b>Baseline</b>	<b>Main</b>	<b>Extend</b>	<b>High</b>	<b>High 10</b>	<b>High 4:2:2</b>	<b>High 4:4:4</b>
4:2:0 Chroma formats	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Monochrome video format (4:0:0)	No	No	No	Yes	Yes	Yes	Yes
4:2:2 Chroma Format	No	No	No	No	No	Yes	Yes
4:4:4 Chroma Format	No	No	No	No	No	No	Yes
8 Bit Sample Bit Depth	Yes	Yes	Yes	Yes	Yes	Yes	Yes
9 and 10 Bit Sample Depth	No	No	No	No	Yes	Yes	Yes
11 to 12 Bit Sample Depth	No	No	No	No	No	No	Yes
8x8 vs. 4x4 transform adaptivity	No	No	No	Yes	Yes	Yes	Yes
Quantization scaling matrices	No	No	No	Yes	Yes	Yes	Yes
Separate C <sub>b</sub> and C <sub>r</sub> QP control	No	No	No	Yes	Yes	Yes	Yes
Residual Color Transform	No	No	No	No	No	No	Yes
Predictive Lossless Coding	No	No	No	No	No	No	Yes
Flexible Macroblock Ordering (FMO)	Yes	Yes	No	No	No	No	No
Arbitrary Slice Ordering (ASO)	Yes	Yes	No	No	No	No	No

## 2.3 Previous Works

In recent years, many researchers proposed a number of optimized algorithms to compute the transforms used in H.264/AVC. The major focus of the research has been to develop fast algorithms for the transform unit.

### 2.3.1 Parallel 4x4 transform and inverse transform Architecture for MPEG-4 AVC/H.264 [5]

The multi-transform approach is good for low power and saving the hardware area. Chen's design [5] is the first multi-transform architecture. They proposed a low power multi-transform architecture. They analyze residuals characteristics and propose a switching power suppression technique for saving data transition power. The design outputs four values every cycle. Their design achieves throughput of eight pixels per cycle and consumes 14.40mW at 200MHz.

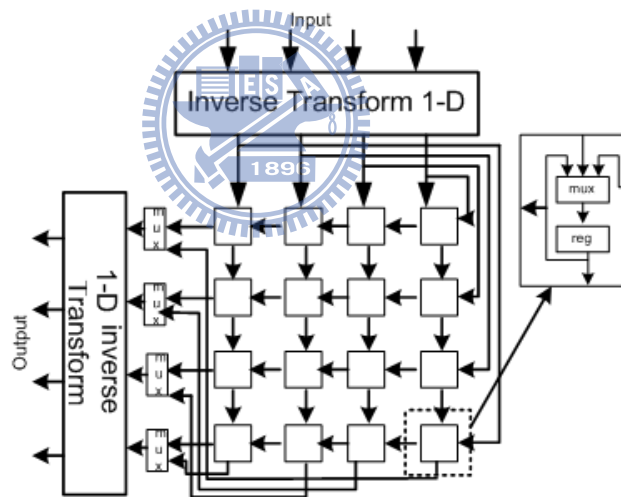


Figure 13. (Re-designed) parallel transform architecture

This architecture is very compact for the 4x4 inverse transform, the gate count is only 4983. The processing speed can be achieved to 1Gpixels/sec at 200MHz. It is sufficient for the existing video formats including HDTV formats. But this architecture is very limited because it can only support 4x4 block. Moreover, if we want to use this architecture and extend to 8x8, it will have almost 4 times overhead. Therefore, this will cost large power consumption and hardware cost. This design still exists some way to accelerate the processing speed and reduce the hardware cost.

### 2.3.2 Low Cost Hardware sharing Architecture of Fast Inverse Transforms for H.264/AVC and AVS Applications [8]

The 1-D fast algorithms and their hardware sharing design for the 1-D inverse transforms of H.264/AVC and AVS are proposed by using the symmetric property of the integer DCT matrix and the matrix decompositions. In this paper hardware-sharing architecture for H.264/AVC and AVS is realized by the offset computations and the pipelined design. Thus, the hardware cost of the proposed sharing architecture for H.264/AVC and AVS is smaller than that of the individual and separate realizations. This design implemented by pipeline stage to increase the performance of inverse transform.

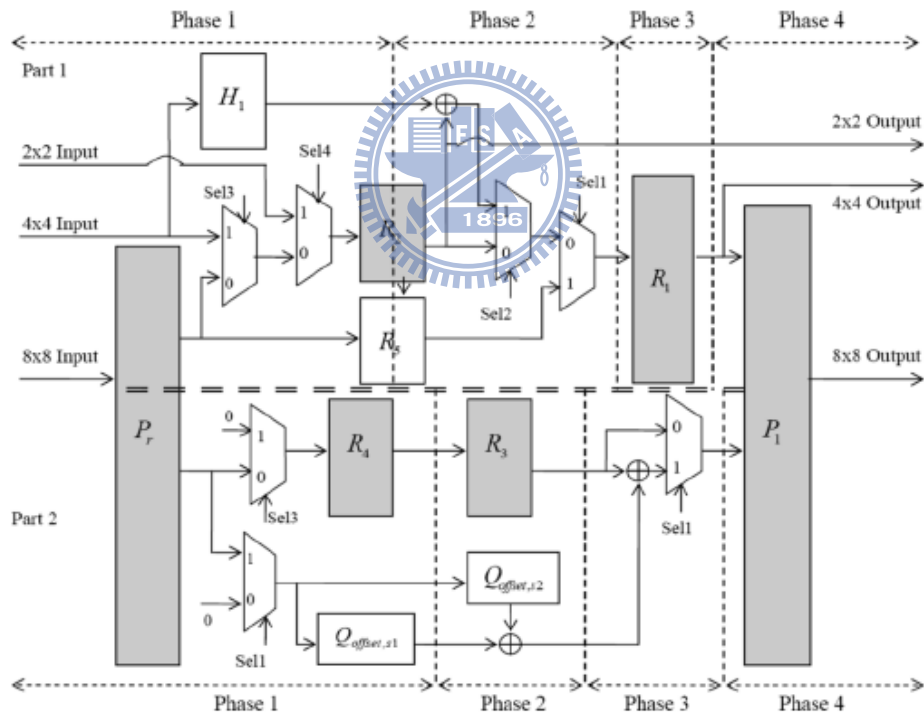


Figure 14. Block diagram of the proposed hardware sharing architecture of fast 2x2, 4x4 and 8x8 inverse transforms for H.264/AVC and AVS with four pipeline phases [8]

In this paper, the 1-D transform is further divided into two smaller matrix-vector operations by even-symmetric or odd-symmetric. Therefore, its size is smaller. But the latency is increased

to 22 cycles because it only consumes one coefficient every cycle. Then the power consumptions of the 8x8 inverse integer at H.264/AVC mode and the 8x8 inverse at AVS mode at 62.5 MHz are 34.266mW and 37.785mW, respectively. Because of the supporting two video standards, need to add extra adding offset computations that use extra registers to completely satisfy two video standards. Therefore the area overhead and power consumption still need to be improved. This design still exists some way to reduce the hardware cost and power consumption.

### 2.3.3 A High Performance Inverse integer Transform Architecture for the H.264/AVC Decoder[12]

In this paper, a high-performance inverse transform architecture for the H.264/AVC decoder is proposed. The proposed architecture utilizes the block multiplication and permutation matrices. This architecture uses the matrix decomposition method to reduce the complexity of 4x4 inverse transform. By applying permutation matrices, the inverse transform matrix is regularized and the inverse Hadamard transform is merged into inverse transform with a minor modification.

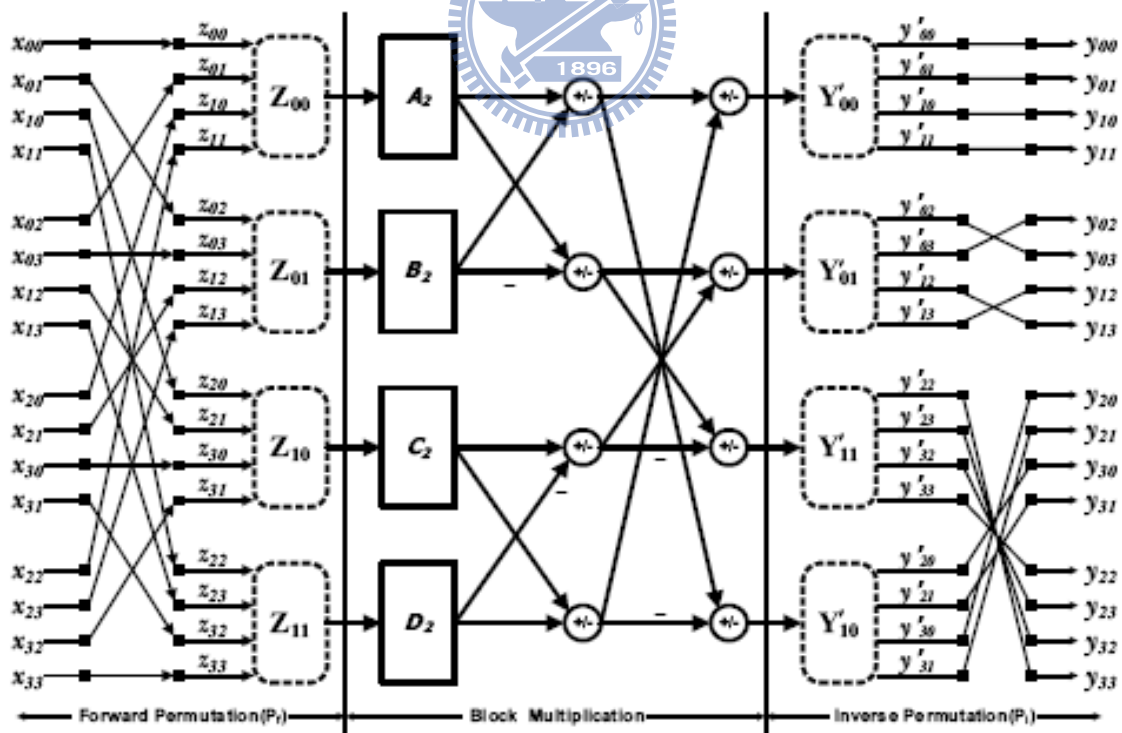


Figure 15. 4x4 inverse transform hardware architecture [12]



This design has higher throughput for computing inverse transform and inverse Hadamard transform. It has also higher hardware efficiency through the measure of DTUA for computing inverse transform and inverse Hadamard transform. In hardware architecture in each block  $A_2$ ,  $B_2$ ,  $C_2$ ,  $D_2$ , they use traditional 4x4 inverse transform algorithm for implementation and too much extra logic was required to completely satisfy H.264/AVC standard. Therefore area and power consumption still need to be improved.

### 2.3.4 Configurable, Low-power Design for Inverse Integer Transform in H.264/AVC[16]

This paper presented a configurable, low-power design for the inverse integer transform in H.264/AVC. The power consumption is drastically reduced by employing an input block-type aware algorithm with variable number of operations for the computation of the inverse integer transform. This algorithm takes advantage of significant number of zero-valued transformed coefficients in a typical input block. Additionally, the area overhead was reduced by designing basic configurable processing blocks in order to share the hardware resources (adders) for different input block types.

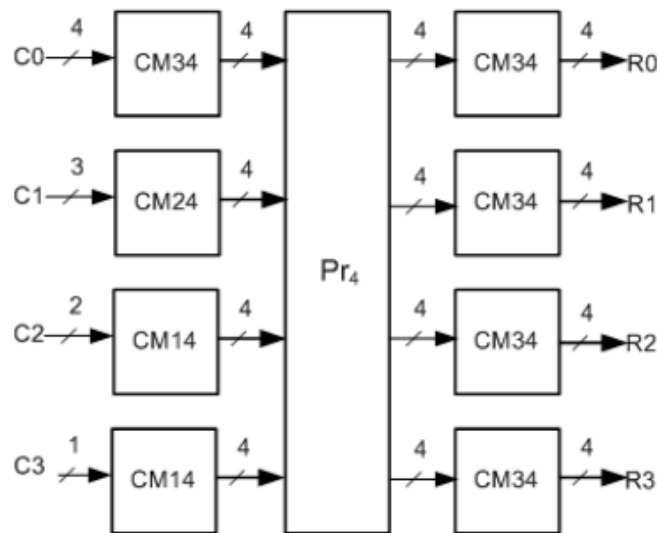


Figure 16. Functional block diagram: Configurable inverse integer transform unit. [16]

The internal organization of this block is depicted in Figure 16. Since the processing block M1-M3 are derived from M4 (Figure 17) and have the similar structure, therefore, we can design a configurable processing units (CM14, CM24, and CM34) with overlapped functionality to reduce the hardware resource requirement for its implementation.

The configurable processing units (CM14, CM24, and CM34) as the name suggest can be configured to provide processing for either (M1, M4), (M2, M4), or (M3, M4) using the appropriate control signal.

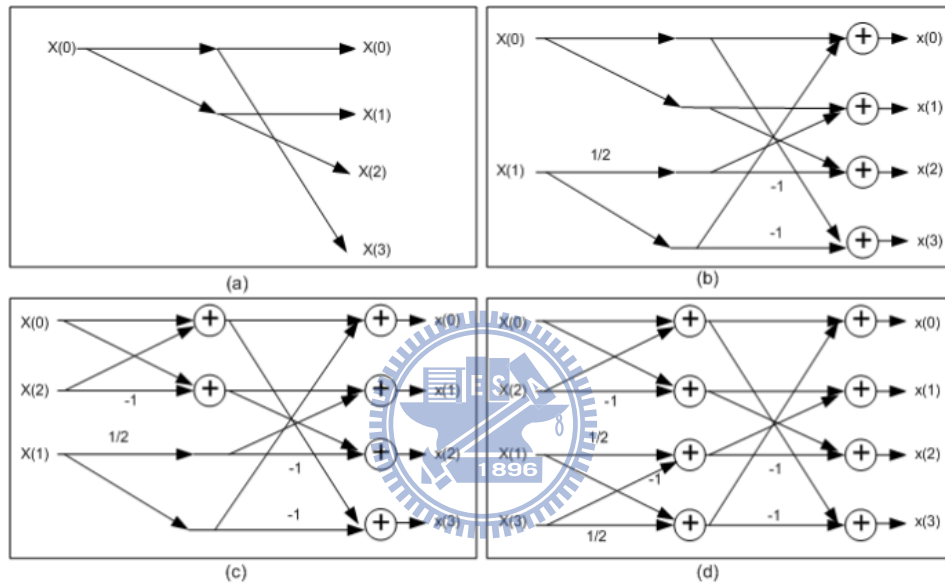


Figure 17. Data flow diagram for (a) M1, (b) M2, (c) M3, and (d) M4 cases.

The internal architecture for these configurable units is depicted in Figure 18(a)-(c). Therefore, no additional (34) adders are required anymore because of configurable processing units. Furthermore, the input registers (in CM24) are also shared among processing for data vectors.

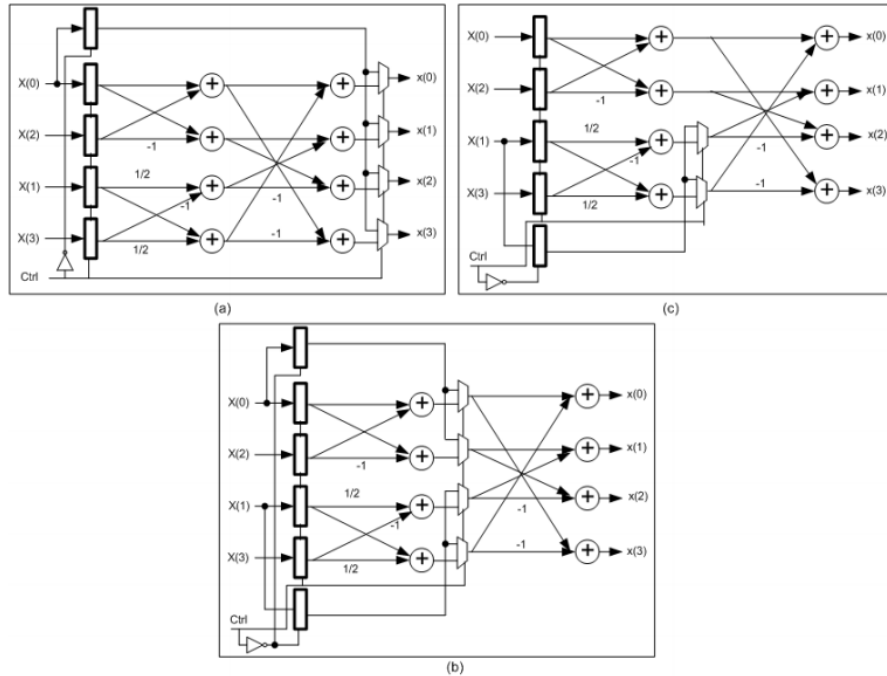


Figure 18. Data flow diagram for (a) CM14, (b) CM24 and (c) CM34.

The new algorithm is derived from the fast one dimensional inverse integer transform. This paper focuses on the low power design that consumes significantly less dynamic power (up to 80% reduction) when compared with existing conventional design for the inverse integer transform. In some blocks, they use traditional 4x4 inverse transform algorithm and this architecture processing speed is very slow that can't achieve the high resolution such as full HD in H.264/AVC.

### 2.3.5 A Reconfigurable IDCT Architecture for Universal Video Decoders[17]

The reconfigurable architecture has become more and more popular. It not only decreases the time of research and development but also saves fabrication cost. Moreover, the proposed reconfigurable inverse integer transform architecture can support 3 different video standards such as VC-1, MPEG and H.264/AVC. The block diagram is shown in Figure 19.

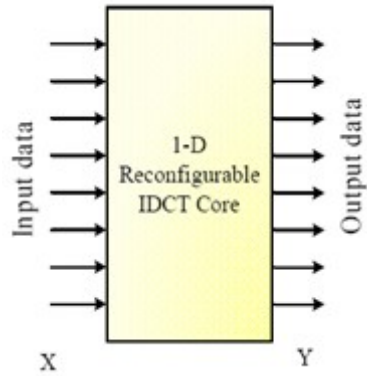


Figure 19. Block diagram of reconfigurable inverse integer transform

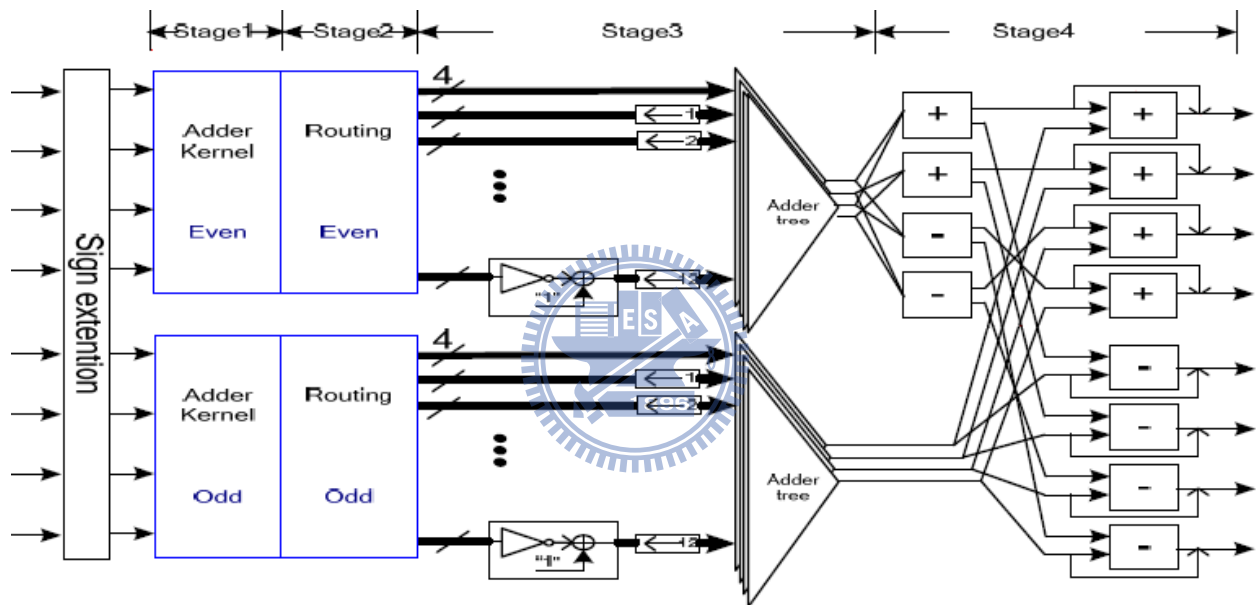


Figure 20. Architecture of reconfigurable one dimensional inverse integer transform

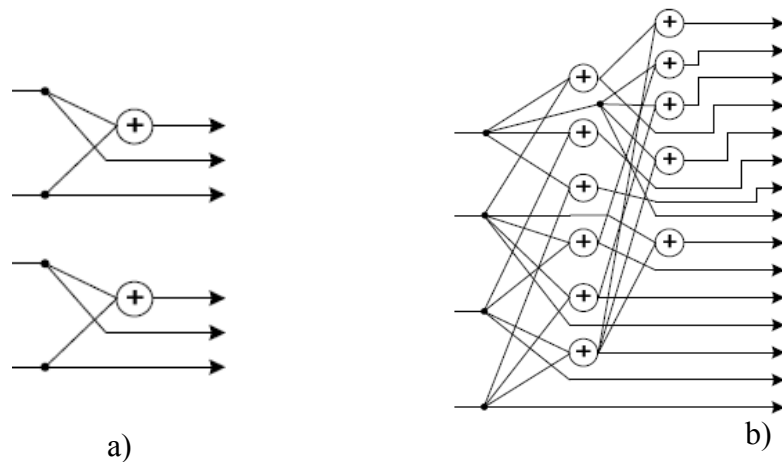


Figure 21. Architecture of adder kernel a) Even part and b) odd part

They propose the reconfigurable one dimensional inverse integer transform architecture combined from two modes in Figure 20 in order to meet the requirements of various video standards. Adder kernel unit, we can find that any combinations of the input signals are composed of  $\{00\sim 11\}$  or  $\{0000\sim 1111\}$ . Therefore the computational results in every row can be generated by adder kernel even and odd part in figure 21. We can simplify the adder kernel into thirteen adders only: two adders in the even parts, figure 21a, and eleven adders in the odd part, figure 21b. Routing network is for VC-1 inverse integer transform. Stage 3 is the shifter and adder tree unit, using two's complement concept to implement the total sums. Stage 4 is the post-adders. Reconfigurable inverse integer transform architecture is implemented for universal video decoders. It is the key point of this paper to reinforce the high throughput and to reduce power consumption and improve the throughput utilizing parallelism. This architecture can support 3 different video standards. The power consumption is 3.4mW at 100MHz, hardware cost is 11.6k and the throughput rate is 800Mpixels/sec. but throughput is still lower than the state of the art such as [7], [10], [12]. In this paper, what kind of fast algorithm that used is not clear and in order to achieve different video standards that use too much extra registers therefore hardware cost still need to be improved.



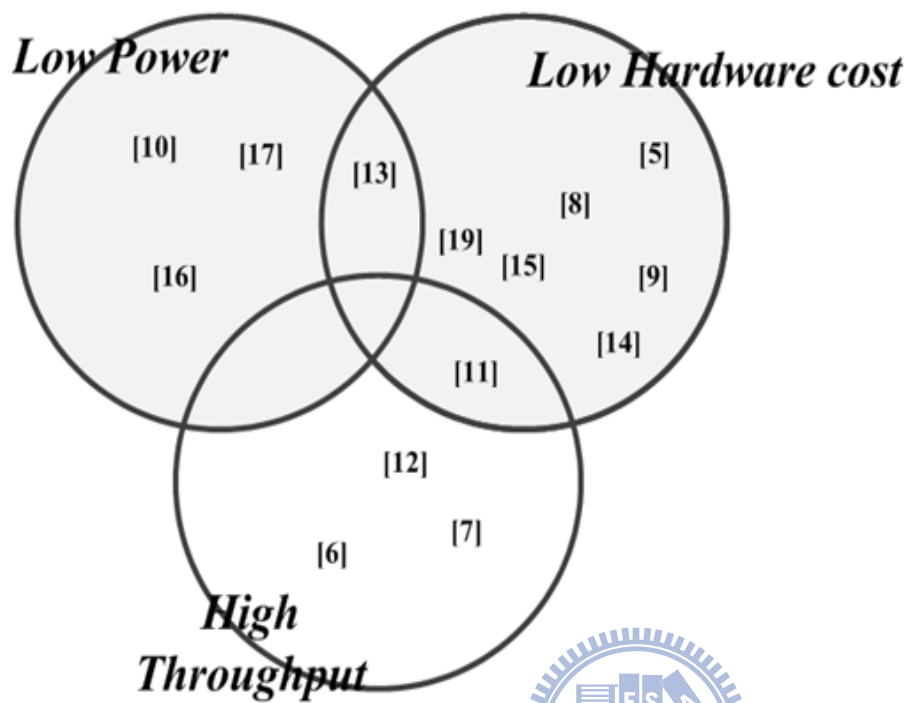
## 2.4 Summary

Table 3 summarized the above approaches. Each has distinct strength and weakness. We take 4x4 transform supporting, 8x8 transform supporting, Hadamard transform supporting, power consumption, hardware cost, DTUA, and throughput as our comparison items.

Table 3. Supporting features comparison

	Hwangbo		Su [8]	Liu [9]	Chen [10]	Cheng [11]	Su [13]	Shia [14]	Lin [15]	Lai [17]
	[7]	[12]								
<b>4x4 transform</b>	Y	Y	Y	Y	Y	Y	N	Y	Y	Y
<b>8x8 transform</b>	Y	N	Y	N	N	N	Y	Y	Y	N
<b>Hadamard</b>	Y	Y	Y	N	Y	Y	N	N	N	Y
<b>low power</b>	N	N	N	N	Y	N	N	N	N	Y
<b>low area</b>	N	N	Y	Y	N	N	Y	Y	Y	N
<b>High DTUA</b>	N	Y	N	N	N	N	N	N	N	N
<b>High Throughput</b>	Y	Y	N	N	N	Y	N	N	N	N

We also take an effort to evaluate several previous works and classify into three strategies: Low power aware, low hardware cost aware and high throughput aware. In Figure 22, we classify previous works as their strategy. Each strategy represents the major improvement in conventional inverse integer transform decoder. Each strategy represents the major improvement in conventional inverse integer transform decoder.



- [5] *ISCAS'03.*
- [6] *TCSVC'09.*
- [7] *TMM'10.*
- [8] *TCSII'08.*
- [9] *IWSOC'04.*
- [10] *VLSI-TSA-DAT'05.*
- [11] *APCCAS'04.*
- [12] *ISCAS'07.*
- [13] *APCCAS'08.*
- [14] *ICME'10.*
- [15] *VLSICAD'06.*
- [16] *FIT'10.*
- [17] *TCE'10.*
- [20] *RSP'08.*



Figure 22. Implementation strategies of previous works

# Chapter 3

## Proposed Algorithm & Architecture

---

In this chapter, we propose our 4x4, Hadamard and 8x8 inverse integer transform fast one dimensional butterfly algorithms, pipeline hardware architectures and in Section 3.4 proposed their hardware-sharing design for 4x4, Hadamard and 8x8 inverse integer transforms of H.264 video decoder. In our algorithms we use matrix decomposition method to reduce the complexity of inverse integer transforms to reduce the power consumption, hardware cost and raise the throughput and hardware efficiency in H.264/AVC. Matrix decomposition utilizes the permutation matrices. All Inverse integer transforms Hardware architecture designs are implemented with pipelined architecture. Thus, our design's power consumption and hardware cost are smaller when comparing to previous works.

The area overhead for the inverse integer transforms unit can be reduced by sharing the hardware resources between the independent processing units by designing the new fast butterfly algorithm. In next sub-sections, we will discuss more details about new fast 4x4, Hadamard, 8x8 butterfly algorithms.

### 3.1 Fast 4x4 Inverse Integer Transform

#### 3.1.1 Fast 4x4 Inverse Integer Transform Algorithm

Fast 4x4 inverse integer transform algorithm is proposed in this part. First we will derive the formulas then algorithms which will be implemented in hardware design. We know that from the previous chapter 4x4 inverse integer transform coefficient matrix (Eq. 3.1) is follows,

$$A_{4_i} = \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \quad \text{Eq. 3.1}$$



We will use the matrix decomposition method to reduce the complexity of inverse integer transform which also means reduce the power consumption, hardware cost in terms of gate count. Therefore we define two permutation matrices  $T_c$  and  $T_r$  as described below,

$$T_c = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad T_r = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Eq. 3.2}$$

And where

$$T_c(T_c)^T = (T_c)^T T_c = I_4, \quad T_r(T_r)^T = (T_r)^T T_r = I_4 \quad \text{Eq. 3.3}$$

$I_4$  is an identity matrix of order 4. By using these permutation matrices then we will get a new matrix  $\tilde{A}_{4i}$ , where the  $\tilde{A}_{4i}$  matrix is described as follows,

$$\tilde{A}_{4i} = (T_c)A_{4i}(T_r)$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

Eq. 3.4

We can re-write the inverse integer transform coefficient matrix from Eq.3.4 matrix multiplication,

$$A_{4i} = (T_c)^T \tilde{A}_{4i} (T_r)^T \quad \text{Eq. 3.5}$$

Then if we can use  $\tilde{A}_{4i}$  matrix to represent with 2x2 matrix form (Eq. 3.6)

$$\tilde{A}_{4i} = \begin{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 1/2 \\ 1/2 & -1 \end{bmatrix} \\ \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & \begin{bmatrix} -1 & -1/2 \\ -1/2 & 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \mathbf{H}_2 & \bar{\mathbf{Q}} \\ \mathbf{H}_2 & -\bar{\mathbf{Q}} \end{bmatrix} \quad \text{Eq. 3.6}$$

Where  $\mathbf{H}_2$  and  $\bar{\mathbf{Q}}$  are

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \bar{\mathbf{Q}} = \begin{bmatrix} 1 & 1/2 \\ 1/2 & -1 \end{bmatrix} \quad \text{Eq. 3.7}$$

Then we use the matrix operation rule to derive one of the following equations (Eq. 3.8);

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{A} & -\mathbf{B} \end{bmatrix} = (\mathbf{H}_2 \otimes \mathbf{I}_2)(\mathbf{A} \oplus \mathbf{B}) \quad \text{Where } \mathbf{I}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{Eq. 3.8}$$

And where  $\otimes$  denotes the Kronecker product, matrix operation as follows (Eq. 3.9). Assume that the dimension of the matrix A is NxP, B is MxQ,

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1P}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2P}\mathbf{B} \\ \vdots & \vdots & \vdots & \vdots \\ a_{N1}\mathbf{B} & a_{N2}\mathbf{B} & \cdots & a_{NP}\mathbf{B} \end{bmatrix}_{MN*PQ} \quad \text{Eq. 3.9}$$

Another  $\oplus$  means the direct sum operation which matrix operation express as follows (Eq. 3.10),

$$\mathbf{A} \oplus \mathbf{B} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{bmatrix}_{2N*2N} \quad \text{Eq. 3.10}$$

If we apply Eq. 3.8 matrix operation rule to Eq. 3.5 then we can re-derive 4x4 inverse integer transform coefficient matrix,  $A_{4i}$  re-expressed as follows,

$$A_{4i} = (T_c)^T \tilde{A}_{4i} (T_r)^T = \mathbf{T}_c^T (\mathbf{H}_2 \otimes \mathbf{I}_2) (\mathbf{H}_2 \oplus \bar{\mathbf{Q}}_2) \mathbf{T}_r^T \quad \text{Eq. 3.11}$$

### 3.1.2 Fast 4x4 Inverse Integer Transform Architecture

According to Eq. 3.11, the first stage of the hardware design  $T_r^T$  and the last stage  $T_c^T$  permutations are just wire connection which represents no arithmetic computation. We use 2 pipeline stages to finish the operation. Figure 23 shows pipeline hardware architecture to represent the 4x4 inverse integer transform and we use the pipeline stages to help us simplify the design and speed up the hardware to achieve the higher resolution such as HD 1080, QFHD(4\*HD 1080) @ 30fps.

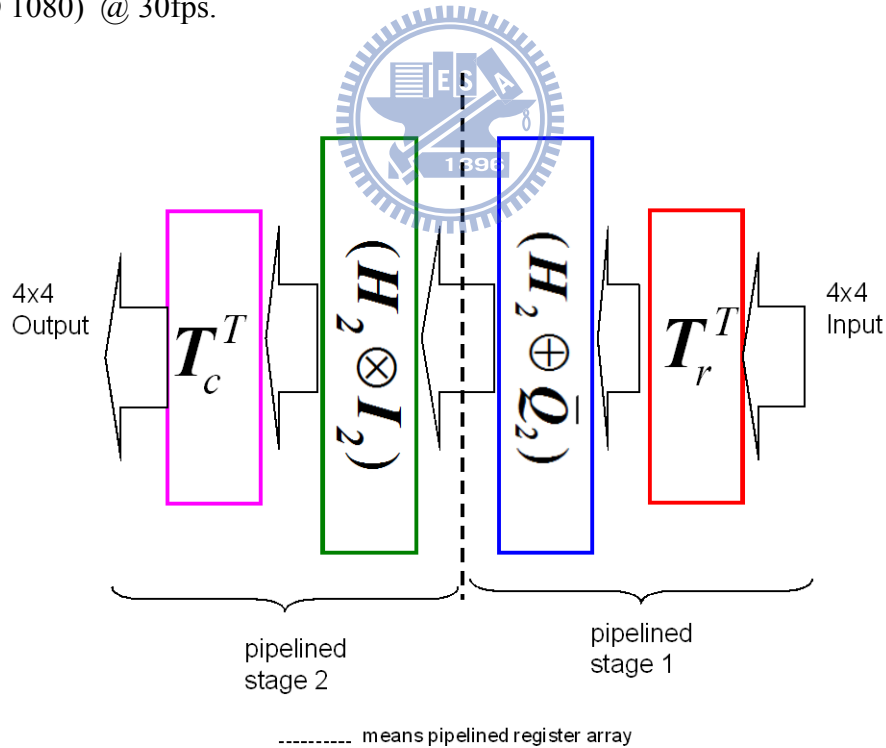


Figure 23. Pipeline hardware architecture for fast 4x4 inverse integer transform.

In pipeline hardware architecture we use fast butterfly algorithm data flow like Figure 24 to implement 4x4 inverse integer transform. The complexity of this proposed fast 4x4 inverse integer transform needs 2 shift operations and 8 additions.

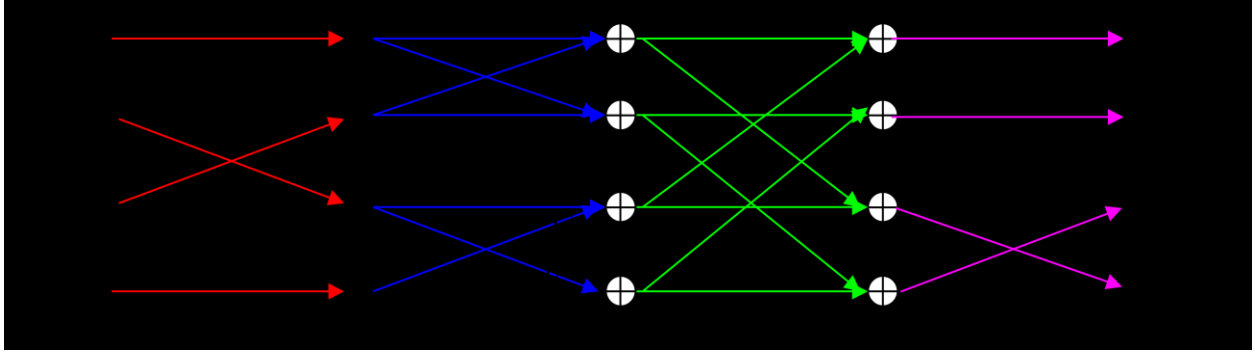


Figure 24. New fast algorithm of 4x4 inverse integer transform.

## 3.2 Fast Inverse Hadamard Integer Transform

### 3.2.1 Inverse Hadamard Integer Transform Algorithm

The Hadamard inverse integer transform is given by:

$$W_D = H_{4i} X_D H_{4i}^T \quad \text{Eq. 3.12}$$

We know that from the previous chapter Hadamard inverse integer transform coefficient matrix,  $H_{4i}$  as follows,

$$H_{4i} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \quad \text{Eq. 3.13}$$

The Hadamard inverse integer transform can be performed in a similar manner. For the Hadamard inverse integer transform, we use the same permutation matrices,  $T_r$  and  $T_c$ ,

$$\mathbf{H}_{4i} = T_c^T \begin{bmatrix} \mathbf{H}_2 & \mathbf{H}_2 \\ \mathbf{H}_2 & -\mathbf{H}_2 \end{bmatrix} T_r^T \quad \text{Eq. 3.14}$$

$$= \mathbf{T}_c^T (\mathbf{H}_2 \otimes \mathbf{I}_2) (\mathbf{H}_2 \oplus \mathbf{H}_2) \mathbf{T}_r^T \quad \text{Eq. 3.15}$$

Where  $\mathbf{H}_2$  is the same matrix given in (Eq. 3.7) before. The first stage of the Hadamard inverse integer transform hardware design  $T_r^T$  and the last stage  $T_c^T$  permutations are just wire connection which represents no arithmetic computation.

### 3.2.2 Inverse Hadamard Integer Transform Hardware Architecture

Figure 25 shows 2 pipeline hardware architectures to represent the 4x4 Hadamard inverse integer transform that we use the pipeline stages to help us simplify the design and speed up the hardware.

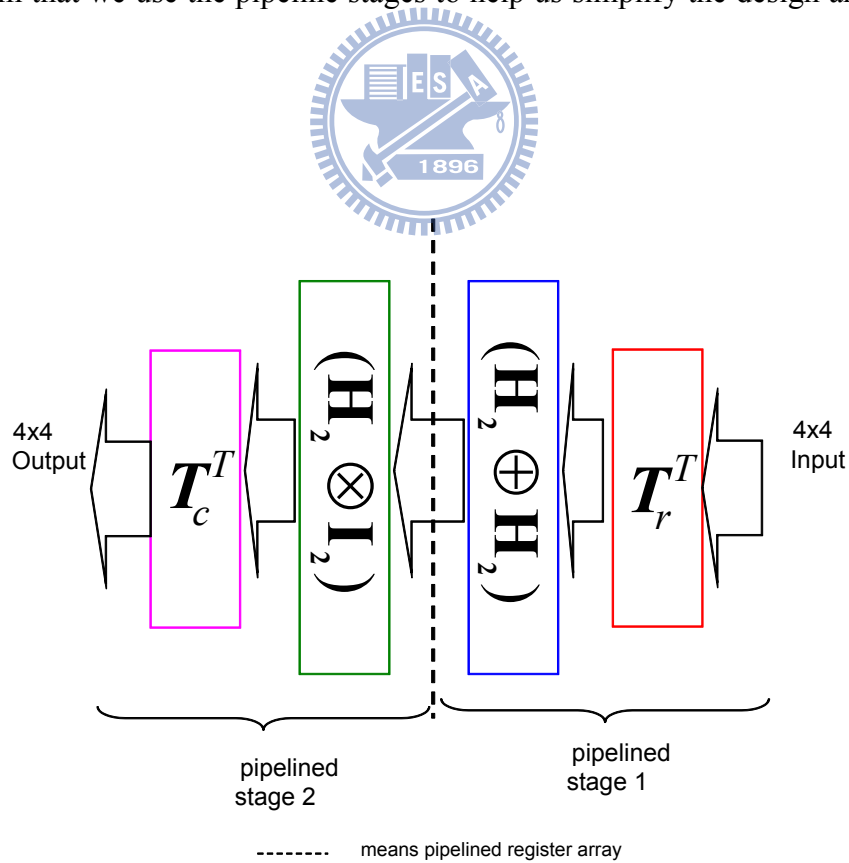


Figure 25. Pipeline hardware architecture for fast Hadamard inverse integer transform.

Figure 26 shows 4x4 Hadamard inverse integer transform implemented by using new fast butterfly algorithm data flow. The complexity of this proposed fast 4x4 Hadamard inverse integer transform needs just 8 additions without any shift operation.

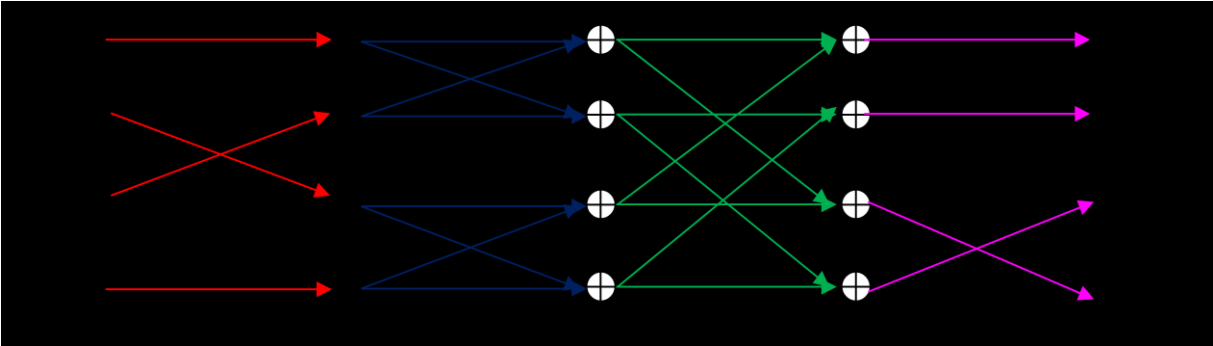


Figure 26. New fast algorithm of 4x4 Hadamard inverse integer transform.



### 3.3 Fast 8x8 Inverse Integer Transform

#### 3.3.1 Fast 8x8 Inverse Integer Transform Algorithm

As we know from the previous chapter 8x8 transform coefficient matrix as follows,

$$C_{8i} = \begin{bmatrix} 8 & 12 & 8 & 10 & 8 & 6 & 4 & 3 \\ 8 & 10 & 4 & -3 & -8 & -12 & -8 & -6 \\ 8 & 6 & -4 & -12 & -8 & 3 & 8 & 10 \\ 8 & 3 & -8 & -6 & 8 & 10 & -4 & -12 \\ 8 & -3 & -8 & 6 & 8 & -10 & -4 & 12 \\ 8 & -6 & -4 & 12 & -8 & -3 & 8 & -10 \\ 8 & -10 & 4 & 3 & -8 & 12 & -8 & 6 \\ 8 & -12 & 8 & -10 & 8 & -6 & 4 & -3 \end{bmatrix} / 8 \quad \text{Eq. 3.16}$$

According to the fast computations in [2], the fast 8x8 inverse integer transform that we use further matrix decomposition into 3 stage matrix multiplication as below,

$$C_{8i} = C_{8i}^1 \cdot C_{8i}^2 \cdot C_{8i}^3 \quad \text{Eq. 3.17}$$

These 3 matrices  $C_{8i}^1$ ,  $C_{8i}^2$ ,  $C_{8i}^3$  are,

$$C_{8i}^1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{bmatrix} \quad \text{Eq. 3.18}$$

$$\mathbf{C}_{8i}^2 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{3}{2} & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & -\frac{3}{2} & 1 \\ 0 & 0 & 1 & 0 & 1 & -\frac{3}{2} & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & \frac{3}{2} \end{bmatrix} \quad \text{Eq. 3.19}$$

$$\mathbf{C}_{8i}^3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} \\ 0 & 0 & 0 & 1 & 0 & -\frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4} & 0 & 1 & 0 & 0 \\ 0 & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \quad \text{Eq. 3.20}$$

Eq. 3.18,  $\mathbf{C}_{8i}^1$  can be further decomposed into,

$$\mathbf{C}_{8i}^1 = \tilde{\mathbf{T}}_c^T \cdot (\mathbf{H}_2 \otimes \mathbf{I}_4) \quad \text{Eq. 3.21}$$



Where  $\mathbf{I}_4$  identity matrix with order 4 and permutation matrix  $\tilde{\mathbf{T}}_c$  is defined by;

$$\mathbf{I}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Eq. 3.22}$$

$$\tilde{\mathbf{T}}_c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad \text{Eq. 3.23}$$

Eq. 3.19,  $\mathbf{C}_{8i}^2$  can be further decomposed into,

$$\mathbf{C}_{8i}^2 = [(\mathbf{I}_2 \oplus \tilde{\mathbf{I}}_2)(\mathbf{H}_2 \otimes \mathbf{I}_2)] \oplus \mathbf{Q}_1^T \quad \text{Eq. 3.24}$$

Where  $\mathbf{I}_2$  identity matrix with order 2 and permutation matrix  $\tilde{\mathbf{I}}_2$  and  $\mathbf{Q}_1$  is equal to as below;

$$\mathbf{I}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \tilde{\mathbf{I}}_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\mathbf{Q}_1 = \begin{bmatrix} \frac{3}{2} & 1 & 1 & 0 \\ 1 & 0 & -\frac{3}{2} & -1 \\ 1 & -\frac{3}{2} & 0 & 1 \\ 0 & 1 & -1 & \frac{3}{2} \end{bmatrix} \quad \text{Eq. 3.25}$$

In Eq. 3.25,  $3x/2$  can be decomposed to  $x+(x>>1)$ . ( $x>>1$ ) means 1 bit right shift. Eq. 3.20,  $\mathbf{C}_{8i}^3$  can be further decomposed into,

$$\mathbf{C}_{8i}^3 = \tilde{\mathbf{C}}_{8i}^3 \cdot \tilde{\mathbf{T}}_r^T \quad \text{Eq. 3.26}$$

Where  $\tilde{\mathbf{T}}_r$  is,

$$\tilde{\mathbf{T}}_r = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Eq. 3.27}$$

And where  $\tilde{\mathbf{C}}_{8i}^3$  can be further decomposed into,

$$\tilde{\mathbf{C}}_{8i}^3 = (\mathbf{H}_2 \oplus \mathbf{Q}_2) \oplus \mathbf{Q}_3^T \quad \text{Eq. 3.28}$$

Where in (Eq. 3.28)  $\mathbf{Q}_2$  and  $\mathbf{Q}_3$ , defined as,

$$\mathbf{Q}_2 = \begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & -1 \end{bmatrix} \quad \text{and} \quad \mathbf{Q}_3 = \begin{bmatrix} 1 & 0 & 0 & \frac{1}{4} \\ 0 & 1 & \frac{1}{4} & 0 \\ 0 & -\frac{1}{4} & 1 & 0 \\ \frac{1}{4} & 0 & 0 & -1 \end{bmatrix} \quad \text{Eq. 3.29}$$

Finally  $\mathbf{C}_{8i}^3$  will be equal to as below;

$$\mathbf{C}_{8i}^3 = [(\mathbf{H}_2 \oplus \mathbf{Q}_2) \oplus \mathbf{Q}_3^T] \cdot \tilde{\mathbf{T}}_r^T \quad \text{Eq. 3.30}$$

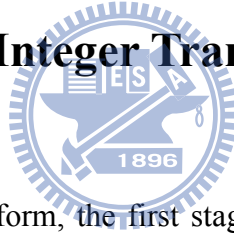
Similarly, in Eq. 3.29,  $x/2$  and  $x/4$  can be replaced by 1 bit right shifter ( $x \gg 1$ ) and 2 bit right shifter ( $x \gg 2$ ) respectively.

Then we can rewrite the 8x8 inverse integer transform matrix,  $\mathbf{C}_{8i}$  can become as follows;

$$\begin{aligned} \mathbf{C}_{8i} &= \mathbf{C}_{8i}^1 \cdot \mathbf{C}_{8i}^2 \cdot \mathbf{C}_{8i}^3 \\ &= \tilde{\mathbf{T}}_c^T \cdot (\mathbf{H}_2 \otimes \mathbf{I}_4) \cdot \{[(\mathbf{I}_2 \oplus \tilde{\mathbf{I}}_2)(\mathbf{H}_2 \otimes \mathbf{I}_2)] \oplus \mathbf{Q}_1^T\} \cdot [(\mathbf{H}_2 \oplus \mathbf{Q}_2) \oplus \mathbf{Q}_3^T] \cdot \tilde{\mathbf{T}}_r^T \end{aligned}$$

Eq. 3.31

### 3.3.2 Fast 8x8 Inverse Integer Transform Hardware Architecture



Same as 4x4 inverse integer transform, the first stage of the 8x8 inverse integer transform hardware design  $\tilde{\mathbf{T}}_r^T$  permutation and the last stage  $\tilde{\mathbf{T}}_c^T$  that need no arithmetic computation to be implemented by hard-wire connection. We use 4 pipeline stages to implement the 8x8 inverse integer transform operation. Figure 27 shows pipeline hardware architecture to represent the 8x8 inverse integer transform. We use the pipeline stages to simplify the design and speed up the hardware to achieve the higher resolution such as HD 1080, QFHD (4\*HD 1080) @ 30fps.

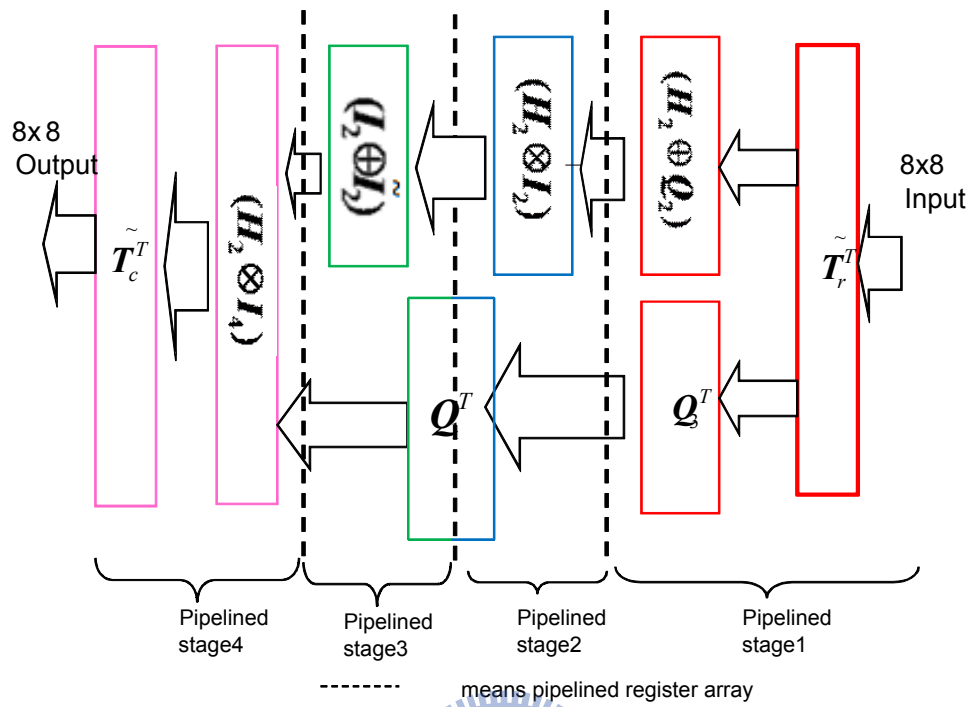


Figure 27. Pipeline hardware architecture for fast 8x8 inverse integer transform.

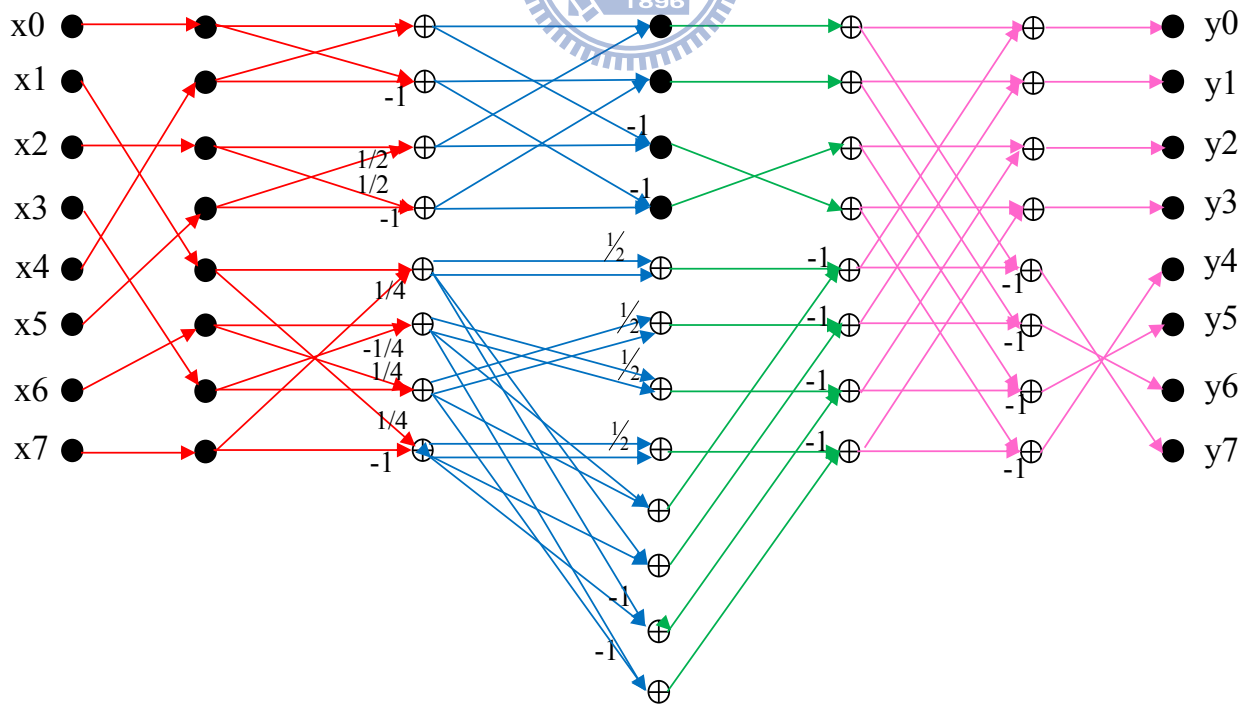
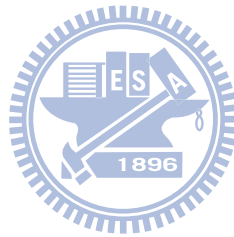


Figure 28. New fast algorithm of 8x8 inverse integer transform.

Figure 28 shows 8x8 inverse integer transform by using new fast butterfly algorithm data flow. Total complexities of proposed fast 8x8 inverse integer transform are just 10 shift and 32 addition operations. In the next section, we will discuss about hardware sharing architecture of inverse integer transform for H.264/AVC decoder.



### 3.4 Hardware Sharing Algorithm & Architecture

First of all, we make all these 4x4 inverse integer transform, Hadamard inverse transform and 8x8 inverse integer transform into one group. It can be found that the inverse transform process is slightly similar. In order to achieve low power and hardware saving for the inverse integer transform unit, sharing the hardware resources between the independent processing units is investigated. For the Hardware sharing design, we have listed all inverse integer transform equations from (Eq. 3.11), (Eq. 3.15), (Eq. 3.31) as follows,

$$\mathbf{A}_{4i} = \mathbf{T}_c^T (\mathbf{H}_2 \otimes \mathbf{I}_2) (\mathbf{H}_2 \oplus \bar{\mathbf{Q}}) \mathbf{T}_r^T$$

$$\mathbf{H}_{4i} = \mathbf{T}_c^T (\mathbf{H}_2 \otimes \mathbf{I}_2) (\mathbf{H}_2 \oplus \mathbf{H}_2) \mathbf{T}_r^T$$

$$\mathbf{C}_{8i} = \tilde{\mathbf{T}}_c^T \cdot (\mathbf{H}_2 \otimes \mathbf{I}_4) \cdot \{[(\mathbf{I}_2 \oplus \tilde{\mathbf{I}}_2)(\mathbf{H}_2 \otimes \mathbf{I}_2)] \oplus \mathbf{Q}_1^T\} \cdot [(\mathbf{H}_2 \oplus \mathbf{Q}_2) \oplus \mathbf{Q}_3^T] \cdot \tilde{\mathbf{T}}_r^T$$

From the above three equations, it can be found that in all these operations have  $(\mathbf{H}_2 \otimes \mathbf{I}_2)$  same blocks. In this block, we will be able to do hardware sharing. Other 3 blocks which are  $\mathbf{H}_2 \oplus \bar{\mathbf{Q}}_2$ ,  $\mathbf{H}_2 \oplus \mathbf{H}_2$  and  $\mathbf{H}_2 \oplus \mathbf{Q}_2$  need to be operated in one hardware block. Since  $\mathbf{Q}_2$  is the main hardware block, for Hadamard we need to use shift in the input circuit (scaling) will meet the  $\mathbf{H}_2$  which is defined in Eq.3.7 in order to save the hardware cost, for 4x4 inverse integer transform that doesn't need any scaling in the input circuit because of  $\mathbf{Q}_2 = \bar{\mathbf{Q}}$  in (Eq. 3.29) and (Eq.3.7). Figure 29 shows the hardware sharing architecture for the fast 4x4, Hadamard, 8x8 inverse integer transforms. The hardware sharing part of the fast inverse integer transforms is  $(\mathbf{H}_2 \otimes \mathbf{I}_2)(\mathbf{H}_2 \oplus \mathbf{Q}_2)$  in Figure 29.

### 3.4.1 Comparison And Implementation Of Hardware Sharing Architecture

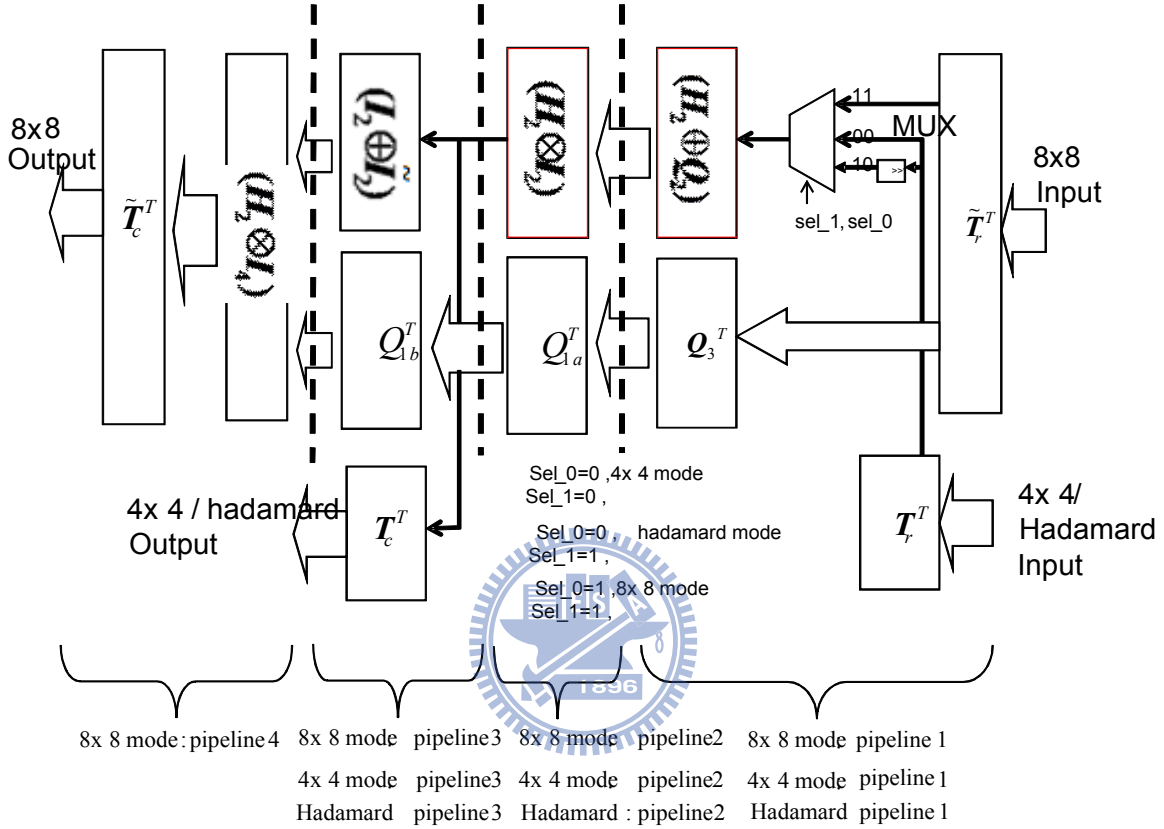


Figure 29. Hardware sharing architecture for fast 8x8, 4x4 and Hadamard inverse integer transform.

In Figure 29, the proposed hardware sharing architectures, which are low power consumption to support 3 inverse integer transform modes, need 12 shifters, 32 additions and a simple MUX in implementations. In order to achieve the purpose of hardware sharing, an additional simple multiplexer is needed for implementation of the fast 4x4, Hadamard, 8x8 inverse integer transform. In Figure 29, the module  $(H_2 \oplus Q_2)$  needs 4 additions and 2 shifters, the module  $(H_2 \oplus I_4)$  needs 8 additions, the module  $(H_2 \oplus I_2)$  needs 4 additions, the module  $Q_3^T$  needs 4 additions and 4 shifters. In Figure 27,  $Q_1^T$  requires 12 additions and 4 shifters. For low power and high processing speed, we cut into two pipeline operation in Figure 29. The first pipeline is

$Q_{1a}^T$  which requires 8 additions and 4 shift registers. The second is  $Q_{1b}^T$  stage pipeline requires just 4 additions. In Table 4, it shows the architecture comparisons for new fast inverse integer transforms. The pipelined phases of hardware sharing architectures are also noted with dotted lines in Figure 29.

Table 4. Architecture Comparisons for Fast Inverse Integer Transforms

Architectures of Inverse Integer Transforms	Number of Shifters	Number of Adders	Supporting Transform Mode
4x4 Only	2	8	4x4
8x8 Only	10	32	8x8
Hadamard only	0	8	Hadamard
Direct Three-mode	12	48	4x4 , 8x8 , Hadamard inverse
Proposed Hardware Sharing Design	12	32	4x4 , 8x8 , 2x2/4x4 Hadamard inverse

The computational complexity of the proposed one dimensional fast 4x4 and 8x8 inverse integer transform is equivalent to that of the fast algorithm of the state-of-the-art [14], where the fast computation needs for 4x4 inverse, 2 shift, 8 addition and for 8x8 inverse 10 shift and 32 addition operations.



### 3.5 Summary and Comparison With Related Works

Based on matrix decomposition algorithm the low power and the most hardware efficiency architecture new fast 4x4, 8x8 and Hadamard inverse integer transforms can be derived. New fast 4x4, Hadamard, 8x8 and hardware sharing inverse transform algorithms and hardware implementations, are developed by utilizing matrix decomposition for H.264/AVC applications. By applying the concept of hardware sharing, the proposed hardware schemes for fast inverse integer transforms need a smaller number of shifters and adders than the direct three mode realization architecture, where the direct architecture just implements the individual 4x4, individual Hadamard, individual 8x8 inverse integer transforms independently (Table 4).

For the throughput, actually we already get high throughput by previous works such as [10] [12]. By the state of the art, we shouldn't keep going to raise much higher throughput. For our purpose, we make an effort to design inverse integer transform decoder which is the most suitable strategy for system integration and take a balance between throughput and overhead at the promise of the acceptable throughput for real-time decoding full-HD sequences. Therefore, we simplify and make a formula for throughput as following:  
First, we can get the formula of total executed cycles,

$$\text{Pixel per cycle (PPC)} = \frac{\text{Total Pixel}}{\text{Total Cycle}}$$

$$\text{Throughput rate} = \text{Inverse of Critical path} \times \text{PPC}$$

In Table 5 also shows the throughput of our designs for 4x4, Hadamard, and 8x8 inverse integer transform and Hardware sharing design.

In our design, we apply pipeline architecture efficiently to get acceptable throughput by our proposed IIT scheme.

On the other hand, we use umc 90nm technology process which is different technology process from previous works such as tsmc 0.18um, umc 0.18um. for the normalization of our works to tsmc 0.18um and umc 0.18um, the normalization result will be the same for tsmc 0.18um and umc 0.18um because of supply voltage is the same 1.8Volt. In order to make fair comparison, in table 5 shows the normalization in terms of the power consumption.

Table 5. Normalization of Power consumption to UMC 0.18um and TSMC 0.18um

	Technology	Processing Unit	Operating frequency(MHz) /Power consumption (mW)
	IIT	IIT	IIT
Gordon[2]'04	N/A	8x8	125MHz / 2.76mW
Chen[10]'05	UMC 0.18um	4x4	100MHz / 9.91mW
Hwangbo[12]'07	UMC 0.18um	4x4	203MHz / N/A
Fan[13]'07	TSMC 0.18um	4x4,8x8	62.5MHz / 2.39mW
Su [8]*'08	TSMC 0.18um	4x4, 8x8, Hadamard,AVS	100MHz / 34.2mW
Su [8]*'09	TSMC 0.18um	8x8	125MHz / 2.79mW
Hwangbo[7]'10	UMC 0.18um	4x4,Hadamard,8x8	200MHz / 86.9mW
Lai [17]'10	UMC 90nm	MPEG, VC, H.264 8x8	100MHz / 3.4mW
Proposed	UMC 90nm	4x4	150MHz / 182.9 $\mu$ W
Proposed		Hadamard	150MHz / 151.8 $\mu$ W
Proposed		8x8	150Mhz / 0.68 mW
Proposed HW Sharing		4x4, Hadamard, 8x8	150MHz / 1.1 mW

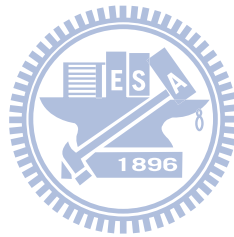
The synthesis results of proposed architecture and the performance comparison with previous works are shown in Table 6. We focus on the power consumption, hardware area and hardware efficiency. Our hardware schemes by applying the concept of hardware sharing for

inverse integer transforms need smaller number of shifters and adders than the direct realization architecture, where the direct realization architecture just implements the individual transforms independently. Through the comparison, the proposed inverse integer transforms design requires, saving more power consumption and performs better hardware efficiency than the state-of-the-art existing design.

Table 6. Synthesis results and Comparison

	Technology	Area (Nand2 Cmos)	Processing Unit	Critical Path Delay (ns)	Operating frequency(MHz) / Power consumption (mW)	Throughput (pixels/sec)	DTUA (Pix./sec/gate)
	IIT	IIT	IIT	IIT	IIT	IIT	IIT
Gordon[2]'04	N/A	6139	8x8	N/A	125MHz / 2.76mW	N/A	N/A
Chen[10]'05	UMC 0.18um	7497	4x4	N/A	100MHz / 9.91mW	2.66G	354.8k
Hwangbo[12]'07	UMC 0.18um	5639	4x4	4.92	203MHz / N/A	3.25G	576.3k
Fan[13]'07	TSMC 0.18um	6.5k	4x4,8x8	N/A	62.5MHz / 2.39mW	N/A	N/A
Su [8]*'08	TSMC 0.18um	9.03k	4x4, 8x8, Hadamard, AVS	N/A	100MHz / 34.2mW	N/A	N/A
Su [8]*'09	TSMC 0.18um	7.03k	8x8	N/A	125MHz / 2.79mW	N/A	N/A
Hwangbo[7]'10	UMC 0.18um	63.6k	4x4, Hadamard, 8x8	N/A	200MHz / 86.9mW	3.2G	50.3k
Lai [17]'10	UMC 90nm	11.6k	MPEG, VC, H.264 8x8	N/A	100MHz / 3.4mW	800M	68.9k
Proposed	UMC 90nm	0.9k	4x4	1.46	150MHz / 56.45 $\mu$ W	2.7G	3M
				0.53	1.49GHz / 405.41 $\mu$ W (Max.)	5.96G	6.6M
0.87k		Hadamard	1.48	150MHz / 46.85 $\mu$ W	2.7G	3.08M	
			0.55	1.53GHz / 401.43 $\mu$ W (Max.)	5.79G	6.5M	
4.2k		8x8	2.11	150Mhz / 0.21 mW	3.8G	904.7k	
			0.64	1.31GHz / 1.46 mW (Max.)	5.12G	1.2G	
Proposed HW Sharing		4.6k	4x4, Hadamard, 8x8	2.23	150MHz / 0.31 mW	3.6G	783.6k
				0.63	625MHz / 1.30 mW (Max.)	5.19	1.1G

In our design, we apply matrix decomposition method to get low power consumption by our 4x4, Hadamard, 8x8 and hardware sharing design scheme. Our hardware architecture power consumption and hardware cost for 4x4, Hadamard, 8x8 inverse integer transform at 56.45 $\mu$ W, 46.85 $\mu$ W, 0.21mW, and for the area 0.9k, 0.87k, 4.2k at 150MHz, respectively. For the Hardware sharing design our power consumption and hardware cost is just 0.31mW and 4.6k, respectively. Our four designs are better power consumption design than the previous works. According to Hardware efficiency index for 4x4, Hadamard, 8x8 and hardware sharing schemes in Table 6, our design is most efficient than existing designs. For the Full HD system speed requirements for each size is 1920x1080 @ 30fps. 4x4, Hadamard, 8x8 and hardware sharing design is suitable for H.264/AVC High Profile.



# Chapter 4

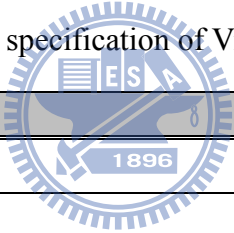
## System Integration

---

### 4.1 System Specification

The specifications of the proposed architecture are described in Table 7 for H.264 video decoder. The proposed architecture is synthesized with UMC 90-nm CMOS standard-cell library and operated at 150MHz. 4x4, 8x8 and 16x16 block size can be supported. Our H.264 decoder, the processing capability for 4x4, Hadamard and 8x8 inverse integer transform are HD1080p/HD720p/QFHD@30fps. In the future trend the higher video resolution is necessary. Therefore our proposed architecture and algorithm can support the higher video resolution than H.264 supported.

Table 7. The specification of Video decoder



<b>H.264/AVC decoder</b>
Process technology : UMC 90nm
Block size: 4x4, 8x8, 16x16
Throughput: 4 – 8 pixels/cycle
Processing capability: 4x4 inverse transform HD1080p,720p, QFHD
Hadamard inverse transform HD1080p,720p, QFHD
8x8 inverse transform HD1080p,720p, QFHD
Decoding capability: H.264/AVC: HDTV, 1080p HD, QFHD @30fps
SVC: 720p– 1080p HD @30fps
Working Frequency:
H.264/AVC: 100 MHz
SVC: 150 MHz

## 4.2 The Integration with H.264/AVC System

In Figure 5, each residual macroblock is transformed, quantized and coded. Previous standards such as MPEG-1, MPEG-2, MPEG-4 and H.263 made use of the 8x8 inverse integer transform as the basic transform. The “baseline” profile of H.264 uses three inverse transforms depending on the type of residual data that is to be coded. A transform for the 4x4 array of luma DC coefficients in intra macroblock (predicted in 16x16 mode), a transform for the 2x2 array of chroma DC coefficients (in any macroblock) and a transform for all other 4x4 blocks in the residual data. If the optional “adaptive block size transform” mode is used, further inverse transforms are chosen depending on the motion compensation block size (4x4, 8x8).

Data within a macroblock are transmitted in the order also shown in Figure 5. If the macroblock is coded in 16x16 Intra mode, then the block labeled “-1” is transmitted first, containing the DC coefficient of each 4x4 luma block. Next, the luma residual blocks 0-15 are transmitted in the order shown (with the DC coefficient set to zero in a 16x16 Intra macroblock). Blocks 16 and 17 contain a 2x2 array of DC coefficients from the Cb and Cr chroma components respectively. Finally, chroma residual blocks 18-25 (with zero DC coefficients) are sent.

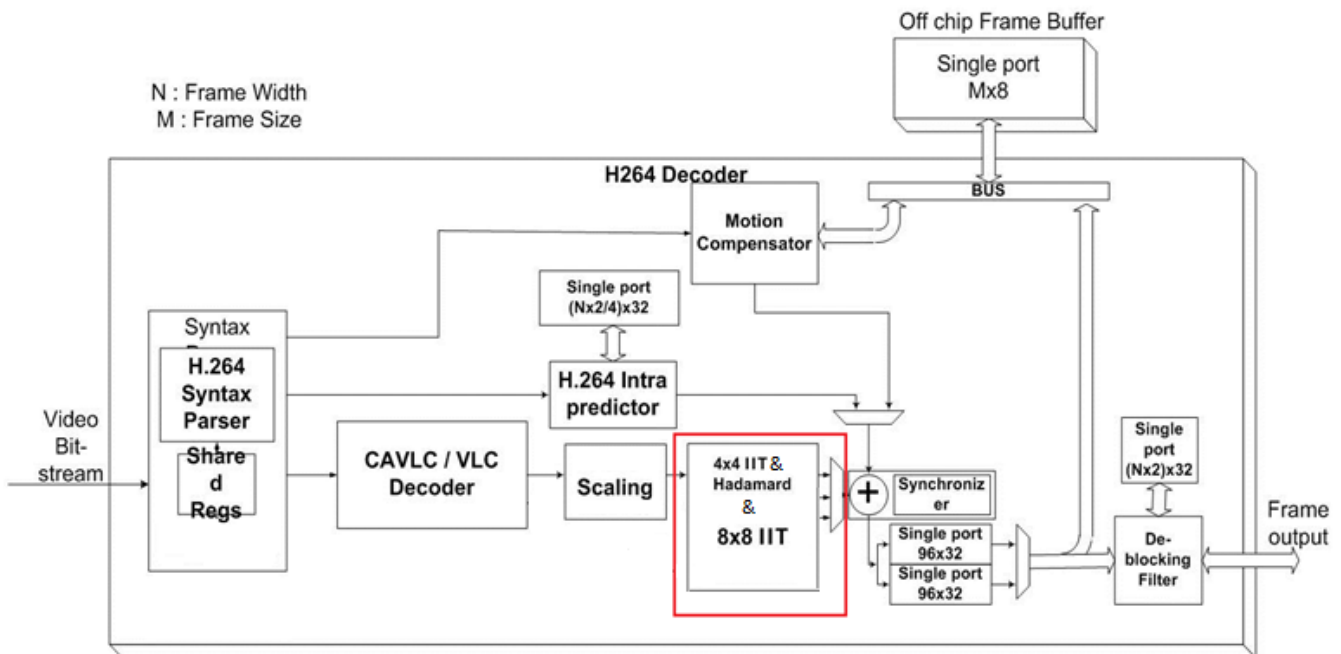


Figure 30. The Integration with H.264/AVC system block diagram

In Figure 30, block diagram shows the integration of inverse integer transform with H.264/AVC video decoder system. Our design architecture can process 4 pixels or 8 pixels per cycle with a low power and small gate count. The input of the inverse transform is from the inverse quantization function followed by a two dimensional 4x4 or 8x8 inverse integer transform depends on the selection mode in hardware sharing design. Then output of the inverse integer transform residual data will be input of the de-blocking filter.

Table 8 shows the timing analysis for different MB. Thus, for the timing analysis, the calculation of time required to process a whole frame is as follows,

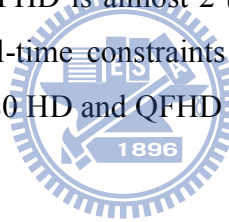
$$\begin{aligned}
 T_{\text{frame}} &= N_{\text{block per frame}} \times T_{\text{block}} \\
 &= \frac{N_{\text{pixel per frame}}}{N_{\text{pixel per block}}} \times T_{\text{block}} \\
 &= \frac{N_{\text{pixel per frame}}}{N_{\text{pixel per block}}} \times N_{\text{cycle}} \times T_{\text{cycle}}
 \end{aligned}
 \tag{Eq. 5.1}$$

Where  $N_{\text{cycle}}$  and  $T_{\text{cycle}}$  indicate the number of cycles and time required per cycle, respectively.

Table 8. Time required to decoding full HD and HDTV frame with different MB. Frame with YUV420 is used.

Frequency	Format	4x4 MB (ms)	Hadamard (ms)	8x8MB (ms)
150MHz	HD 1080 (1920x1080)	4.53	4.6	5.4
	HD 720 (1280x768)	2.14	2.18	3.1
625MHz	QFHD (4*HD 1080)	17.3	17.3	18

$T_{HD\_4x4MB}$  (4.53ms) is 7.35,  $T_{HD\_Hadamard}$  (4.6ms) is 7.23,  $T_{HD\_8x8MB}$  (5.4ms) is 6.1 times faster than the 33.3ms standard time required for processing each HD frame decoding. Same way for the HDTV frame,  $T_{HDTV\_4x4MB}$  (2.14ms) is 15.5,  $T_{HDTV\_Hadamard}$  (2.18ms) is 15.27,  $T_{HDTV\_8x8MB}$  (3.1ms) is 10.7 times faster and for QFHD is almost 2 times faster. Thus, the proposed inverse transforms architectures meet the real-time constraints for HD1080 and QFHD video signal. Therefore this module can perform 1080 HD and QFHD @ 30fps in real-time.





# Chapter 5

## Conclusion and Future Works

---

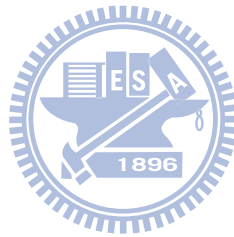
### 5.1 Conclusion

In this works, we implement 4x4, Hadamard, 8x8 inverse integer transforms and Hardware sharing design. We first proposed fast algorithm for 4x4 and 8x8 macroblock and use with pipeline to reduce the inverse transform complexity which means saved power consumption, significant reduce hardware area and enhance the performance of the hardware. Our hardware architecture power consumption and hardware cost for 4x4, Hadamard, and 8x8 inverse integer transforms are only  $56.45\mu\text{W}$ ,  $46.85\mu\text{W}$ , and  $0.21\text{mW}$  at 150MHz and for the area 0.9k, 0.87k, 4.2k, respectively. For the Hardware sharing design our power consumption is just  $0.31\text{mW}$  and hardware cost is just 4.6k. Our four designs are better power consumption design than the previous works. For the Full HD system speed requirements for each size is  $1920 \times 1080 @ 30\text{fps}$ .

Our comparisons power consumption, hardware cost in terms of gate count, critical path delay, throughput and hardware efficiency which achieves better (783.6k) than the previous works. DTUA is used to evaluate the hardware efficiency. It is defined as the ratio of data throughput rate over hardware cost in terms of the gate count. The higher the DTUA is, the more efficient the design. According to the DTUA in Table 6, our four designs are the most hardware efficient design than other designs. In Table 6, the proposed hardware sharing design for fast 4x4, Hadamard, 8x8 inverse transforms of H.264/AVC requires smaller gate counts (i.e., 4.602 gates) than the individual 4x4 and 8x8 inverse integer transforms without the hardware share (i.e.,  $904+873+4209=5986$  gates). This component can be used in H.264 high profile decoder design and its inversion can be used in encoder design as well.

## 5.2 Future Work

In the future, we will more focus on new algorithms to reduce the number of adder and shifter that saving more power consumption and keep improving the performance and to further reduce hardware area of our design. We will also employ voltage scaling technique to further reduce power consumption and furthermore employ gated clock and multiple clock technique to save the clock power. Meanwhile, we will try to support other standard inverse transforms in the same algorithms.



## References

- [1] Joint Video Team (JVT) of ISO/IEC MPEG&ITU-T VCEG, “*Joint Draft ITU-T Rec. H.264 | ISO/IEC 14496-10 Scalable video coding*,” July 2007.
- [2] S.Gordon, D.Marple, and T. Wiegand, “Simplified use of 8x8 Transforms – Update Proposal and results,” *JVT-K028, 11<sup>th</sup> Meeting*, Munich, Germany, 15-19, Mar. 2004.
- [3] Iain E. G. Richardson, *H264 and MPEG-4 Video Compression-Video Coding for Next-generation Multimedia*, John Wiley & Sons Ltd, 2003.
- [4] D.Marpe, T. Wiegand, and S. Gordon, “H.264/MPEG4-AVC fidelity range extensions: Tools, profiles, performance, and application areas,” *IEEE International Conference of Image Processing*, pp. I-593-I-596, Sep. 2005.
- [5] T. C. Wang *et al.*, “Parallel 4x4 2D transform and inverse transform architecture for MPEG-4 AVC/H.264,” *IEEE International Symposium on Circuits and Systems*, pp.800-803, May 2003.
- [6] C. P. Fan “Efficient Fast 1-D 8x8 Inverse Integer Transform for VC-1 Application,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol.19, no.4, pp.584-590, April 2009.
- [7] W. Hwangbo, J. Kim and C.M. Kyung, “A Multi Transform Architecture for H.264/AVC High-Profile Coders,” *IEEE international transactions on multimedia*, Vol. 12, No.3, pp.157-167, Apr. 2010.
- [8] G. A. Su, “Low-Cost Hardware Sharing Architecture of Fast 1-D Inverse Transforms for H.264/AVC and AVS Applications,” *IEEE Transactions on Circuits and Systems, Part II*, vol.55, no.12, pp.1249-1253, Dec. 2008.
- [9] L. Z. Liu *et al.*, “A 2-D forward/inverse integer transform processor of H.264 based on highly-parallel architecture,” *IEEE International Workshop on System-on Chip for Real-Time Applications*, pp.158-161, July 2004.

- [10] K. H. Chen, J. I. Guo, *et al.*, “A high-performance low power direct 2-D transform coding IP design for MPEG-4 AVC/H.264 with a switching power suppression technique,” *IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test*, pp.291-294, Apr. 2005.
- [11] Z. Y. Cheng *et al.*, “High throughput 2-D transform architectures for H.264 advanced video coders,” *IEEE Asia-Pacific Conference on Circuits and Systems*, pp.1141-1144, Dec. 2004.
- [12] W. Hwangbo, J. Kim and C.M Kyung, “A High-Performance 2-D Inverse Transform Architecture for the H.264/AVC Decoder,” *IEEE International Symposium of Circuits and Systems, 2007. ISCAS 2007*, pp.1613-1616, May 2007.
- [13] G.A. Su *et al.*, “Cost Effective Hardware Sharing Architecture for Fast 1-D 8x8 Forward and Inverse Integer Transforms of H.264/AVC High Profile,” *IEEE Asia Pacific Conference of Circuits and Systems, 2008. APCCAS 2008*, pp.1332-1335, 2008.
- [14] M.L. Hsia, and T.C.C. Oskal, “Low-complexity inverse integer transform in H.264/AVC,” *International Conference of Multimedia and Expo (ICME)*, pp.826-830, 2010.
- [15] Y.K Lin, Y.Z Liao, and T.S. Chang “An area-efficient Design for Integer Transform in H.264/AVC FRExt,” *The 17<sup>th</sup> VLSI Design/CAD symposium*, 2006.
- [16] M.Nadeem *et al.*, “Configurable, Low Power Design for Inverse integer Transform in H.264/AVC,” *8<sup>th</sup> International Conference on Frontiers of Information Technology (FIT)*, no.8, Dec. 2010.
- [17] Y. K. Lai, and Y. F. Lai, “A Reconfigurable IDCT Architecture for Universal Video Decoders,” *IEEE Transactions on Consumer Electronics*, vol.56, no.3, pp.1872-1879, August 2010.
- [18] H.S. Malvar, A. Hallapuro, M. Karaczewicz, 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, July 2003.

- [19] N.T. Ngo, T.T.T. Do, T.M. Le, “ASIP-controlled Inverse Integer Transform for H.264/AVC Compression”, The 19th IEEE/IFIP International Symposium, pp.158-164, June 2008.
- [20] C. P. Fan, and Y. L. Cheng, “Unified and Fast 2-Dimension 4x4 Transform Design for H.264/AVC Texture Coding”, IEEE International Symposium on Intelligent Signal Processing and Communication Systems, pp.473-476, December 2005.
- [21] R. A. Horn and C. R. Johnson, *Topics in Matrix Analysis*. New York: Cambridge Univ. Press, 1991, pp. 239-267.

