

# ***Chapter 2***

## ***Joint MPEG-2 and H.264/AVC Decoder***

---

### *2.1 Background*

Multimedia raises some exceptionally interesting topics concerning interoperability. The most obvious issue concerning multimedia interoperability relates to format incompatibility. For example, prevalent MPEG standards are backward compatible. But, the advent of H.264/AVC and VC-1 cannot be backward compatible to the former H.26x and MPEG-x families of video coding standards. Moreover, in an application point of view, digital video broadcasting (DVB) project has paved the way for the introduction of MPEG-2 based digital TV service, known as DVB-T in many countries. Recently, DVB-H, a spin-off of the DVB-T standard, adopts the transmission of H.264/AVC for handheld digital TV due to its bandwidth-efficiency. DVB-H is totally backward compatible to DVB-T but is transmitted with different video contents (i.e. MPEG-2 vs. H.264/AVC). In other words, a generic problem of standard-incompatibility has emerged, resulting in the design challenge for the multimedia interoperability.

Such a wealth of available standards inevitably produces incompatibility problem, commonly solved by directly combining or transcoding from one format to another. However, transcoding from one standard to another faces drifting error due to the mismatch of motion compensation, and needs additional hardware to re-encode and decode bit-stream, leading to problems of coding latency and processing power. Instead, a direct integration for different standards can be easily achieved without aforementioned problems but increases design cost. Moreover, we only consider decoding side and there is no need to transcode or

recompress one stream to another format. In this chapter, we aim at a combining or integrating method so as to improve hardware utilization and choose a well-known MPEG-2 and newly-announced H.264/AVC as our decoding platform to support both video standards.

## 2.2 Overview

H.264/AVC aims at providing functionality similar to prevalent MPEG-2, but with significantly better coding performance. The improved performance comes from some new techniques such as spatial prediction in intra coding, adaptive block-size motion compensation, 4×4 integer transformation, context-adaptive entropy coding, and adaptive deblocking filtering. To integrate these new techniques of H.264/AVC into MPEG-2, a combined data flow is designed: it is composed of the residual path and predicted path shown in Figure 2.1. In the residual path, a context-adaptive variable length decoder (CAVLD) or VLD translates the received streams into symbols through a table-lookup method. The follow-up processes first reorder the symbols into a 2-dimensional block using inverse zig-zag (I-ZZ) scan, rescale (inverse quantization, or I-Q) the frequency-domain coefficients of a block, and then perform inverse discrete cosine transform (IDCT) to produce residual pixels. On the other hand, the macroblock type *mb\_type* can be decoded by an MPEG-2/H.264 syntax parser and is defined to select the source of predicted pixels. These pixels come from either spatially predicted (intra prediction) or temporally predicted (motion compensation) blocks. The addition of predicted and residual blocks will be performed in a pixel-wise manner. Afterward, the filtered results are sent into external memory through synchronous DRAM (SDRAM) interface (I/F) for decoding subsequent frames. Moreover, a separate data path is utilized for onscreen display (OSD) through the display I/F. The results of the display engine are sent into the display monitor, in either

digital (CCIR656) or analog form.

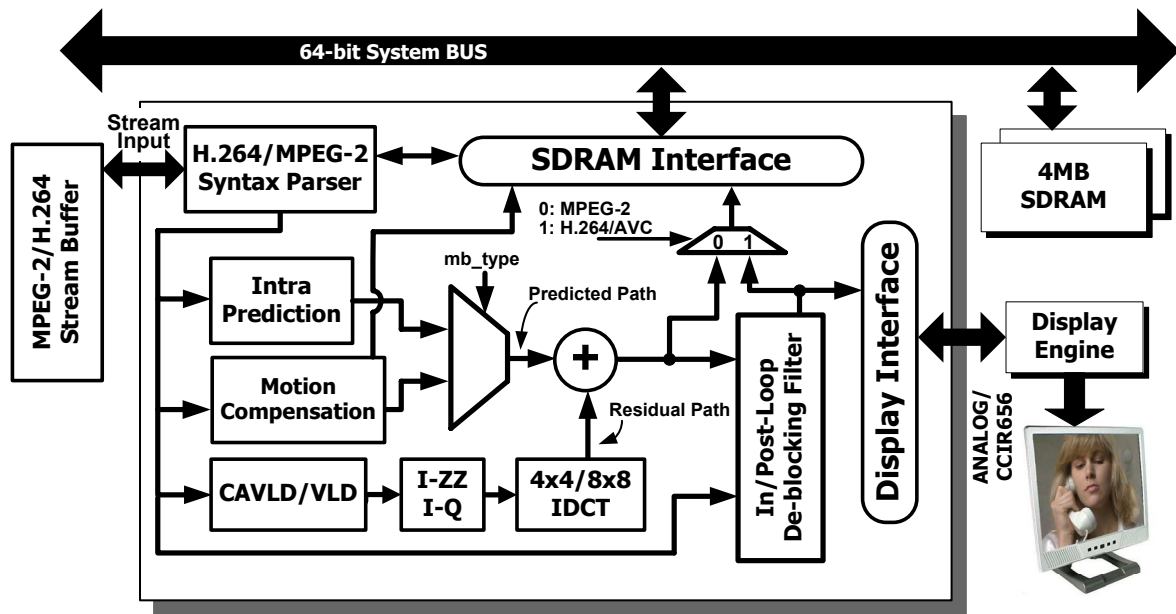


Figure 2.1: System block diagram.

Table 2.1 lists the similarity of each module. First, we implement a custom-built syntax parser and exploit a register sharing technique to reduce register numbers. Second, one codeword cannot be the prefix of another codeword in a table but this rule does not hold among different standards. Hence, most of VLC code-words can be merged in both standards. Third, intra prediction in MPEG-2 is just a sub-set of that in H.264/AVC since H.264/AVC features to employ the multi-directional prediction to improve coding efficiency. Fourth, motion compensations in both standards intend to perform interpolation procedures. Several adders and multipliers can be combined by applying resource sharing techniques. Although the aforementioned modules improve the hardware utilization, the inverse transforms between MPEG-2 and H.264/AVC are so diverse that they are difficult to combine. Similarly, the integration of deblocking filters has the same problem as well. In the following, we will go into the details in each functional block to address how to integrate both standards in an area-efficient manner.

Table 2.1: The similarity analysis of each key module.

Key Module		MPEG-2	H.264/AVC	Similarity
Syntax Parser		Register-rich	Register-rich	Register-rich
Entropy Decoder		VLD	Context-adaptive VLD	variable length table
Intra Prediction		Frequency DC prediction	Directional spatial prediction	predictive coding
Inverse DCT		8x8 cosine kernel	4x4 integer kernel	
Motion Compensation	Luma	Bilinear	Half: 6-tap FIR Quarter: 6-tap FIR/Bilinear	bilinear interpolation
	Chroma	Bilinear	Bilinear	
Deblocking Filter		N/A (user-defined)	In-loop adaptive filter	

### 2.3 Syntax Parser

First, syntax parser decodes the header information from a bitstream and provides several control signals to subsequent modules. It behaves as a finite state machine and is a register-rich unit. To efficiently share registers in the syntax parser of different standards, we implement a custom-built syntax parser instead of a platform-based solution. In general, registers for parameter storage and control circuits are two main components in a syntax parser design. Because video streams require a large amount of headers to parse the correct data, registers almost dominate the area as well as cost in the syntax parser. Considering a dual-standard integration, a video playback is realized by executing either MPEG-2 decoding or H.264/AVC decoding. That is, there is always a set of syntax parser in an idle mode. Hence, we can share commonly-used registers in both standards for a cost-saving design approach. Table 2.2 shows the required registers in both standards. Because MPEG-2

is coded on an  $8 \times 8$  block level, 4 times bigger than the  $4 \times 4$  sub-block coding in H.264/AVC, the numbers of required registers in MPEG-2 are more than that in H.264/AVC. We share registers in three main modules (i.e. *Slice\_header*, *PictureParameterSet*, *SequenceParameterSet*) of H.264/AVC with MPEG-2. Therefore, Experimental results reveal that the numbers of registers can be reduced by 26% compared to a separate design.

Table 2.2: Number of register needed for MPEG-2/H.264 syntax parser

MPEG-2		H.264/AVC	
Module	# of Registers	Module	# of Register
Macroblock	29 regs	Macroblock	11 regs
Motion_vectors	25 regs	Slice_data	17 regs
Picture_coding_ext	49 regs	Slice_header	247 regs
Picture_header	36 regs	PictureParameterSet	74 regs
Slice	8 regs	SequenceParameterSet	127 regs
Sequence_header	1105 regs		
Total	1252 regs	Total	476 regs

## 2.4 Context-Adaptive Variable Length Decoder

The entropy coding in MPEG-2 and H.264/AVC is variable length coding (VLC) and context-adaptive variable length coding (CAVLC), respectively. Although the coding flows are widely different, both coding methods are based on variable length tables and can be merged to reduce the table size. The concept of VLC is to assigns shorter codewords to more frequent symbols, and vice versa. As for CAVLC, VLC tables for various elements are selected depending on previously coded coefficients. It results in the improvement of coding efficiency as compared to the traditional method that uses a single VLC table. Although

those VLC tables are defined in different standards, we can share those tables since each table will be enabled at different time slices. Hence, we develop a table-merging method (cf. [24]) to combine those tables in each standard or in different standards.

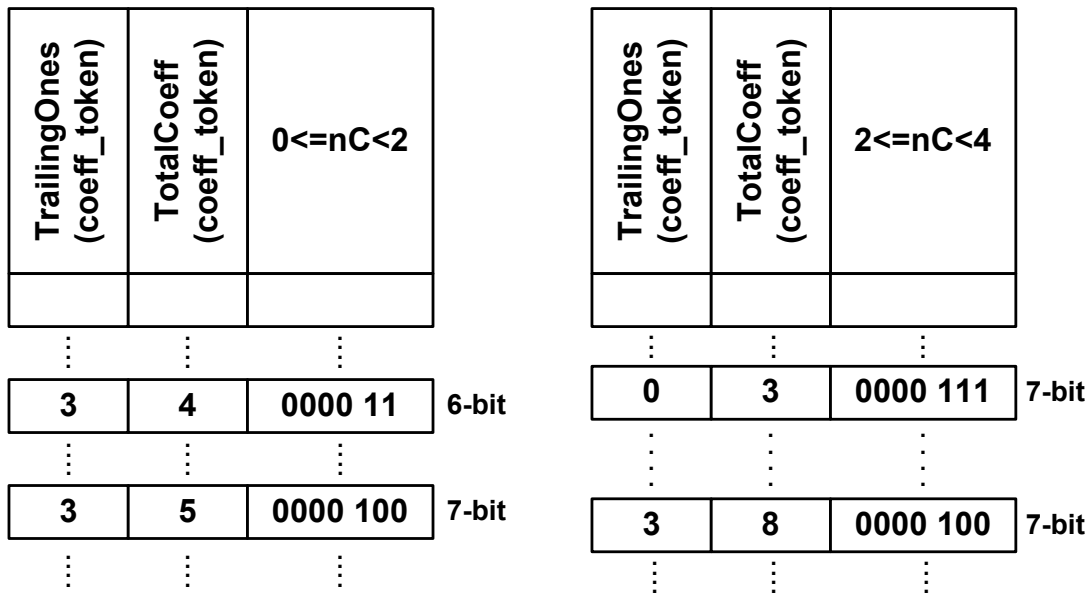
### 2.4.1 Table-Merging Method

The table-merging method is developed to merge different tables which are defined in one standard or different standards. There are two methods developed to merge tables. One is the codeword-merging and the other is the prefix-merging method. In the codeword-merging method, although most VLC coding tables are generated based on the Huffman procedure, one codeword still has high probability to exist in many coding tables. If this case occurs, it is unnecessary to duplicate the codeword information in memories for every table that uses this codeword. A codeword-merging method is applied to set this codeword as a merged codeword and reuse the codeword information when the coding tables are required. Therefore, the information redundancy among coding tables is exploited. The stored data are reduced from many identical codewords to one merged codeword. As for the prefix-merging method, according to the Huffman property, one codeword cannot be the prefix of another codeword in a table but this rule does not hold among different tables. Frequently, a short codeword in one table will be the prefix of a long codeword in other tables. When these codewords are found, a prefix-merging method is undertaken by storing the long codeword as a merged VLC codeword and the lengths of the VLC codewords in each table. As a result, the information redundancy among tables is further exploited.

To help the understanding of this merged process, Figure 2.2 demonstrates the table-merging method through the CAVLC table that maps *coeff\_token* to *TotalCoeff* and *TrailingOnes*. Figure 2.2(a) and (b) are two VLC tables and 27-bit (6+7+7+7) are required for storing codeword information. By applying prefix-merging and codeword-merging

methods, the required space for merged table is reduced to 22-bit in Figure 2.2(c). In addition to codeword information, additional table information, which is to recover VLC coding tables from merged table, is stored since it's hard to distinguish which table is used to generate a merged codeword. Hence, every VLC code-length of all tables has to be stored individually and will not be reused even though table-merging method is applied. To further suppress the code-length information, we exploit differential code-length instead. It represents the distance between the code-length of merged and individual codewords. Moreover, to select the merged codeword of VLC tables quickly, a valid bit is utilized to indicate whether a merged codeword belongs to the table. Therefore, in Figure 2.2(c), the additional table information is required to facilitate the decoding process and the merged table size is reduced from 27 to 22 bits.

A table merging process is accomplished by applying both codeword-merging and prefix-merging methods to the codewords of all AC transform coefficient (TCOEF) tables in MPEG-2 and five VLC tables (i.e. *Run\_Before*, *Total\_Zeros*, *Levels*, *Trailing1\_Sign*, and *Coeff-Token*) in H.264/AVC. Considering those tables, decoded symbols have been stored into memory and cannot be combined. If we only consider the number of bits in codewords, there are 370 and 759 bits in MPEG-2 VLC and H.264/AVC CAVLC coding tables, respectively. After applying the table merging method, total numbers of codewords have been reduced to 936 bits which is approximately 80% of a separate design ( $370+759 = 1,129$  bits).



(a)

(b)

Merged codeword	$0 \leq nC < 2$		$2 \leq nC < 4$		
	Valid	$D_{CL}$	Valid	$D_{CL}$	
0000 111	1	1	1	0	11-bit
0000 100	1	0	1	0	11-bit

Prefix-Merging → (points to the 0000 111 row)  
Codeword-Merging → (points to the 0000 100 row)

(c)

Figure 2.2: (a)(b) VLC tables in CAVLC and (c) the merged table.

## 2.5 Intra Prediction

Intra prediction is a well-known method to predict the pixel value based on values previously coded in one frame. This prediction can be carried out in either spatial or temporal domain. Prevalent MPEG-2 exploits intra prediction in the frequency domain such as DC prediction after the DCT transformation while H.264/AVC performs intra prediction in the spatial domain prior to the DCT. Both standards require subtraction operation for predictive coding, and those operations can be shared without any structural conflict.



However, considering the integration between MPEG-2 and H.264/AVC, the percentage of resource sharing is considerably low because the hardware cost in the intra prediction of MPEG-2 is far less than that of H.264/AVC. On the other hand, the intra prediction can be not only integrated among different standards but also combined with inter prediction in H.264/AVC video standards. This combined intra/inter prediction engine can be realized because prediction scheme will be enabled in either intra or inter predicted manner. Hence, we address how to achieve this integration between intra and inter prediction (i.e. motion compensation) in H.264/AVC.

### 2.5.1 Combined Intra/Inter Prediction

This sub-section demonstrates a new architecture for combining intra and inter predictions in H.264/AVC video decoder. This architecture is proposed by Li *et al.* [25] for a cost reduction approach. As we know, H.264/AVC simultaneously incorporates inter and intra predictions to remove temporal and spatial redundancy. Both predictions require intensive FIR filtering processes and can share operating elements since each macroblock is coded in either inter or intra mode. In addition, because the prediction mode of each macroblock is known in advance, a combined architecture achieves better hardware utilization compared with the separate design. Figure 2.3 gives an overview of this combined architecture. We address how to implement the combined method in an H.264/AVC decoding flow and didn't go into the details of the combined FIR filter [25]. As shown, in inter prediction mode, the input comes from the motion compensated buffer. Herein, we assume that the data is already transferred from frame memory to local buffer. On the other hand, the data is inputted from a row store buffer that keeps boundary pixels in adjacent block when intra prediction mode is applied. Li *et al.* [25] also highlights this design contribution by comparing with other leading-edge approaches where the intra and

inter predictions are realized separately. Considering the number of adders, experimental results reveal that 22~88% of cost reduction can be achieved.

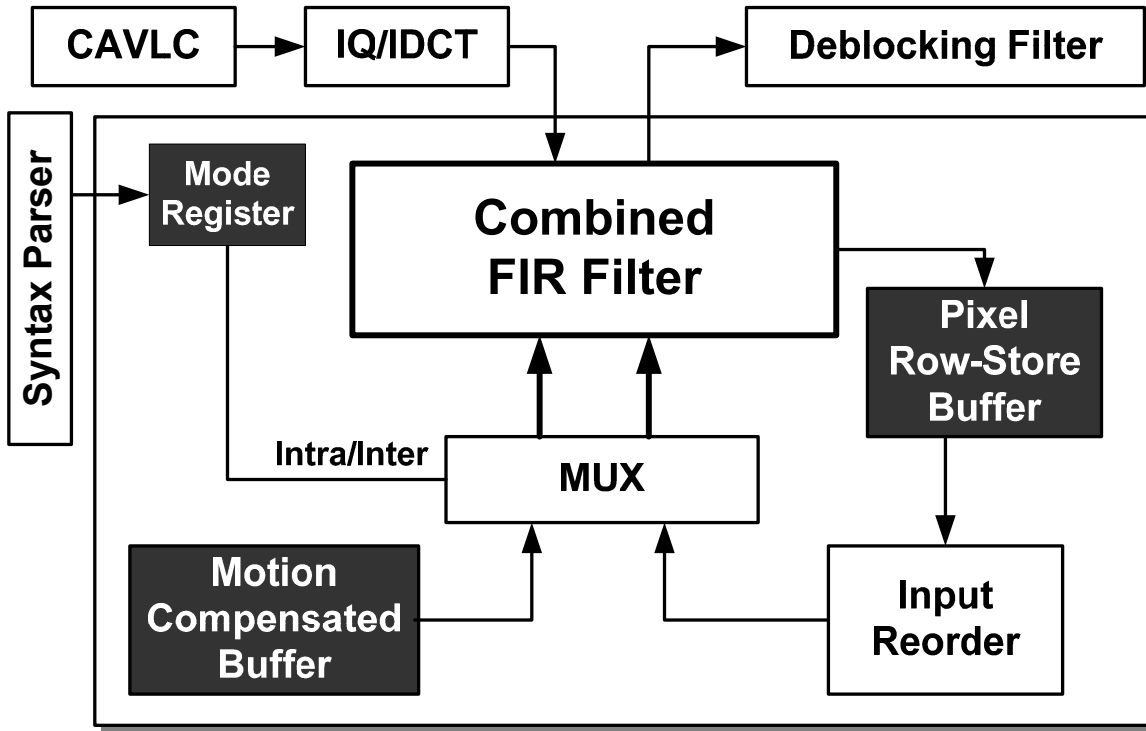
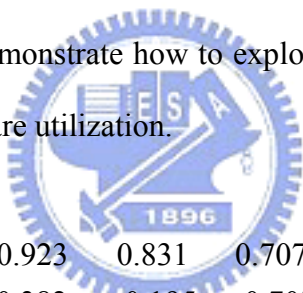


Figure 2.3: Overview of combined intra and inter predictions.

## 2.6 Inverse Discrete Cosine Transform

A major focus in integrating MPEG-2 and H.264/AVC is IDCT since it faces the most diverse algorithms over the whole design. As shown in Figure 2.4, the IDCT kernel of H.264/AVC is a  $4 \times 4$  integer transform kernel, but that of MPEG-2 is an  $8 \times 8$  cosine transform kernel. Due to an algorithmic difference with respect to transform size and kernel characteristics, a shared IDCT structure presents a great challenge and existing solutions usually contain two individual IDCT modules without sharing. Although a recent work, Park *et al.* [26], presents a flexible transform processor for multi-CODECs, they only combine transpose memories without efficiently integrating different transform kernels. In Figure 2.5, we exploit two 8-point IDCTs for row and column transforms, respectively, and

an 8×8 pixel buffer for matrix transposition. Furthermore, considering the data path in IDCT, the specifications of H.264/AVC and MPEG-2 guarantee that 16-bit and 12-bit arithmetic are enough respectively. As for the combining issue, by allocating the lowest required bit width to data paths and registers, the accuracy of each functional unit is 16-bit and thereby meets IEEE std. 1180-1990 requirement [27]. Each data bus requires four paths and stands for four-pixel (i.e. 4×16-bit) operation where the dotted lines realize the 4×4 IDCT in H.264/AVC. We make use of a recursive algorithm to extract the lower orders of transformation matrix for solving the problem of different transform size. Hence, the 8×8 IDCT in MPEG-2 can also be performed in a 4×4 fashion and one-fourth of pixel buffers are shared for different IDCT operations. Moreover, we develop a multiple constant multiplier (MCM) structure to efficiently share several operation units (e.g. additions and shifts). In the following, we demonstrate how to exploit the recursive algorithm and MCM structure to improve the hardware utilization.



$$H_{8 \times 8} = \begin{bmatrix} 0.707 & 0.980 & 0.923 & 0.831 & 0.707 & 0.555 & 0.382 & 0.195 \\ 0.707 & 0.831 & 0.382 & -0.195 & -0.707 & -0.980 & -0.923 & -0.555 \\ 0.707 & 0.555 & -0.382 & -0.980 & -0.707 & 0.195 & 0.923 & 0.831 \\ 0.707 & 0.195 & -0.923 & -0.555 & 0.707 & 0.831 & -0.382 & -0.980 \\ 0.707 & -0.195 & -0.923 & 0.555 & 0.707 & -0.831 & -0.382 & 0.980 \\ 0.707 & -0.555 & -0.382 & 0.980 & -0.707 & -0.195 & 0.923 & -0.831 \\ 0.707 & -0.831 & 0.382 & 0.195 & -0.707 & 0.980 & -0.923 & 0.555 \\ 0.707 & -0.980 & 0.912 & 0.831 & 0.707 & -0.555 & 0.382 & -0.195 \end{bmatrix}$$

(a)

$$H_{4 \times 4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix}$$

(b)

Figure 2.4: Two inverse kernels for (a) MPEG-2 and (b) H.264/AVC.

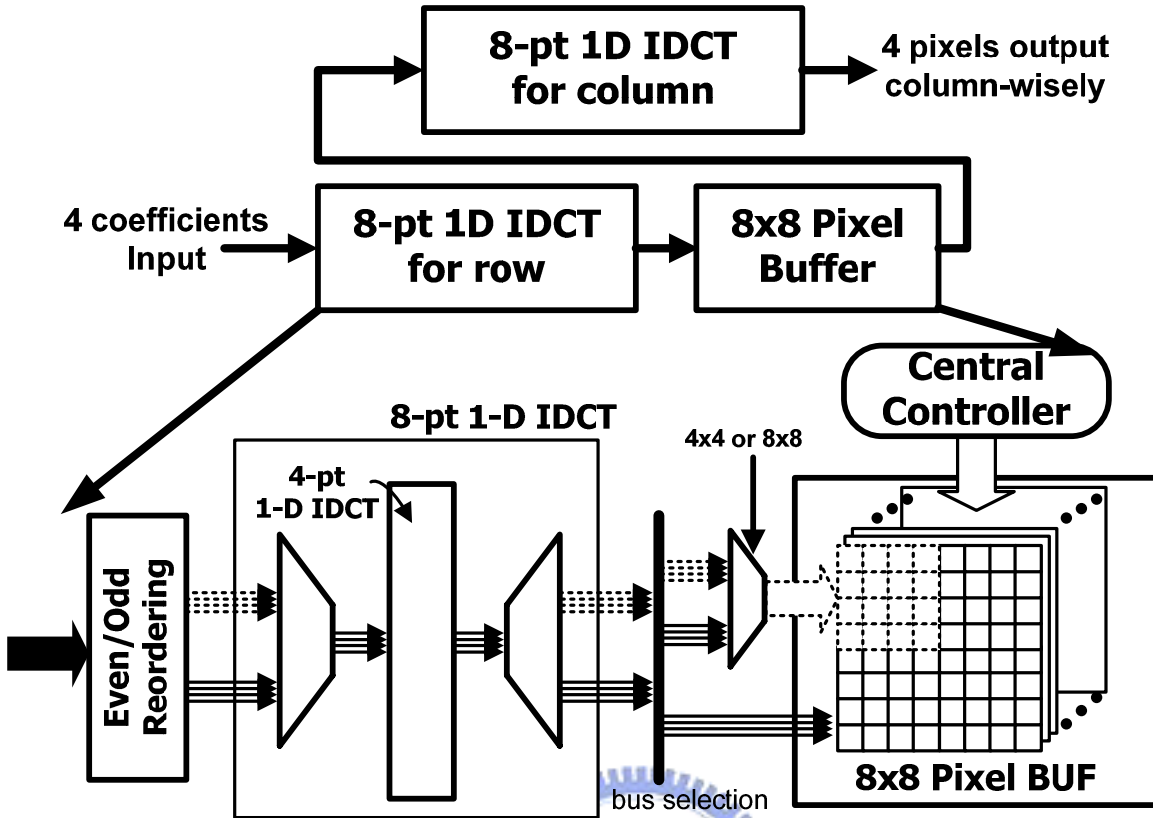


Figure 2.5: 4x4/8x8 IDCT core block diagram.

### 2.6.1 Recursive Algorithm

To efficiently integrate 8x8 transform size into 4x4, a recursive algorithm was preliminarily presented by Hsieh S. Hou in 1987 [28] and exploited to improve the hardware utilization in algorithmic levels. In this algorithm, the 8-point IDCT can be computed using 4-point IDCT recursively. In other words, N-point IDCT can be decomposed into an N/2-point IDCT by reordering even and odd coefficients and selectively storing IDCT results into pixel buffers. This allows us to generate 8-point IDCT from lower-order 4-point IDCTs [28]. Therefore, this 4-point IDCT can be simultaneously shared both in MPEG-2 and H.264/AVC, and the problem of different transform size can be resolved. We show the partitioned process of MPEG-2 8x8 IDCT in the following equation (N=8).

$$z(a, n) = \sqrt{\frac{2}{N}} \sum_{a=0}^{\frac{N}{2}-1} \alpha(a) Y(2a) \cos\left(\frac{(2n+1)\pi(2a)}{2N}\right) + \sqrt{\frac{2}{N}} \sum_{a=0}^{\frac{N}{2}-1} \alpha(a) Y(2a) \cos\left(\frac{(2n+1)\pi(2a+1)}{2N}\right)$$

$$\alpha(0) = \sqrt{\frac{1}{2}} \text{ and } \alpha(k) = 1, k \neq 0, n, k = 0, 1, \dots, N-1$$

## 2.6.2 Multiple Constant Multiplication

MPEG-2 8×8 kernel features a floating point operation while H.264/AVC 4×4 kernel is an integer operation. Although 8×8 IDCT intends to use multipliers to realize inverse transforms, we replace these multiplications with a series of shifts and additions. In particular, we exploit the common sub-expression sharing techniques for solving multiple constant multiplication (MCM) problems [29]. That is, we can expand constant multiplications with shift operations and additions and share the common ones. Therefore, these operating units can be shared with 4×4 IDCT to reduce silicon area. Moreover, because both standards require addition and the input bit-width of adders in H.264/AVC is smaller than those in MPEG-2, the common terms of addition can be reused between MPEG-2 and H.264/AVC. Therefore, 8×8 IDCT is carried out in a 4×4 fashion, and one-fourth of pipelined buffer can be reused in different standards. Finally, this proposal saves 15% gate-count than the one without exploiting any hardware sharing.

## 2.7 Motion Compensation

When considering motion compensation between MPEG-2 and H.264/AVC, we notice two major differences: prediction size and interpolated resolution. The prediction size in MPEG-2 is fixed to 16×16 pixels while H.264/AVC supports variable block size from 16×16 to 4×4 (i.e. 16×16, 8×16, 16×8, 8×8, 4×8, 8×4, 4×4). Hence, the block size in MPEG-2 can be considered as a sub-set of that in H.264/AVC. On the other hand, the interpolated resolution in MPEG-2 and H.264/AVC is up to half-pel (i.e. 1/2) and eighth-pel

(i.e. 1/8) respectively. Hence, we can conclude that motion compensation in H.264/AVC outperforms that in MPEG-2 because H.264/AVC has variable block sizes and finer predictive resolutions. Moreover, motion compensation in MPEG-2 is just a sub-set of that in H.264/AVC and can be easily merged. In the following, we will address the register-sharing in interpolator design for solving the integration issue.

### 2.7.1 Hybrid MPEG-2/H.264 Interpolator Design

A major challenge of combined motion compensation engine is interpolator. In this sub-section, we will focus on storage and arithmetic module sharing to minimize area/cost overhead. For macroblock-based fractional motion compensation in MPEG-2, each  $16 \times 16$  macroblock needs  $17 \times 17$  interpolation windows to interpolate fractional samples. Each macroblock can be partitioned into four  $8 \times 8$  blocks with  $9 \times 9$  interpolation window of which size is identical to that of H.264/AVC luma interpolation window for each  $4 \times 4$  sub-block. Additionally, in Table 2.1, the bilinear filter for H.264/AVC luma quarter-pel interpolation can be shared with that for MPEG-2 half-pel interpolation. Considering a separate 1-D luma interpolator [30] in Figure 2.6, the content buffer represents storage to execute a content-swap operation in one cycle and part of registers and bilinear filters, which are shaded, can be shared with MPEG-2's interpolator.

In addition to the integration between MPEG-2 and H.264/AVC, the luma and chroma interpolator of H.264/AVC can be shared as well since chroma interpolation processes are carried out after luma interpolation. Figure 2.7 shows the combined interpolator design. Specifically, both luma and chroma interpolators have similar interpolation processes and require a great number of addition operations. Hence, the adders can be reused for different interpolation processes. As a result, based on 0.18- $\mu\text{m}$  technology, the proposal reduced the gate count by 20% as compared to a separate interpolator design under a working frequency

of 100MHz.

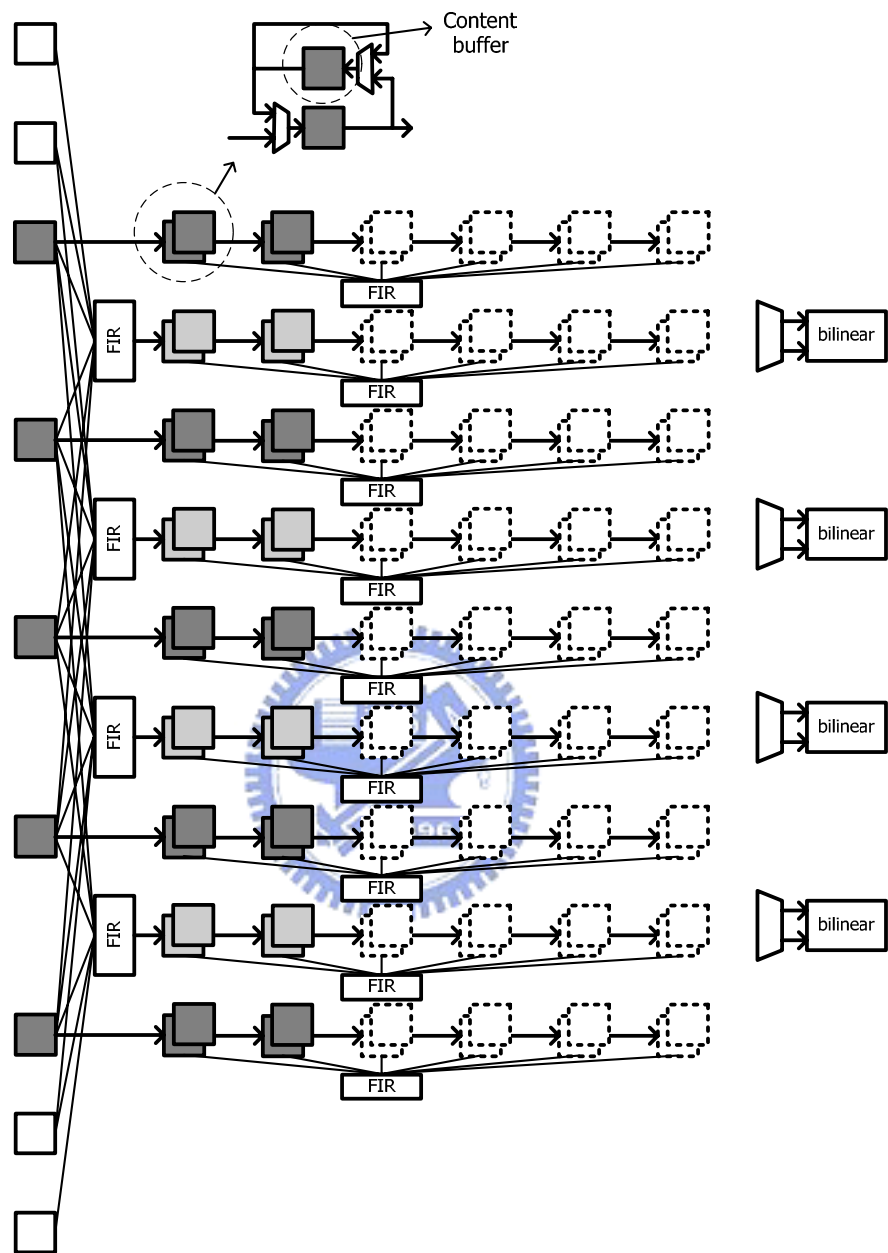


Figure 2.6: Shared local registers and bilinear filters for MPEG-2 and H.264/AVC.

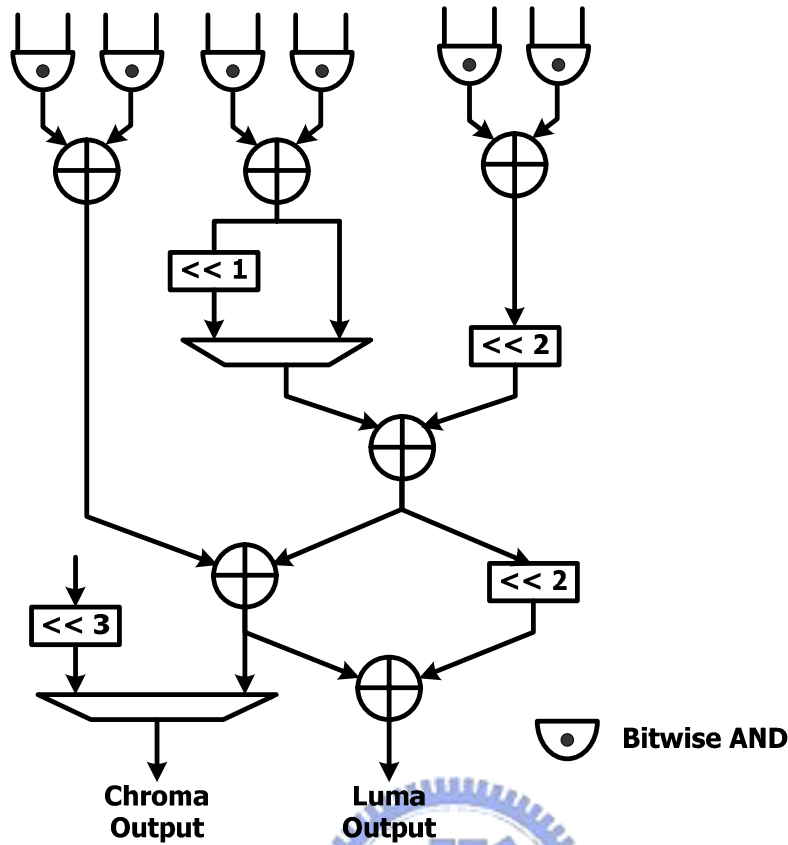


Figure 2.7: Combined luma/chroma interpolator design in H.264/AVC.

## 2.8 Deblocking Filter

### 2.8.1 Background

Deblocking schemes can be divided into two classes: in-loop and post-loop de-blocking filters. An in-loop filter is standardized by H.264/AVC and a post-loop filter follows the prevalent MPEG-x family standards. Hence, the goal of this research is to integrate in-loop and post-loop filter in a hardware-sharing fashion. Figure 2.8 outlines existing deblocking filters in terms of filtering types and standardization. In general, post-loop de-blocking filters [32][33] are executed outside the DPCM loop. They need extra frame buffer to store filtered frame and induce the issues of computation for VLSI implementation. In contrast, the in-loop de-blocking filter [31] operates inside the loop and thus is normative in the standardization process. It feeds through the filtered results into the



reference frames for subsequent frame decoding. The PSNR performance of in-loop filter outperforms that of post-loop filter but offers a significant complexity due to its high adaptivity and smaller filtering size ( $4 \times 4$ ). Moreover, considering the integration issue, although existing solutions [35][36] propose a unique algorithm to meet both in-loop and post-loop filters, those algorithms are not adopted by existing video standards and perform the post-loop filter with extra frame buffer. To alleviate aforementioned problems, we derive a new algorithm that can be reconfigured as an in-loop or post-loop filtering processes. Specifically, we propose an in/post-loop algorithm to easily meet different standard requirements. Due to the non-standardization of post-loop filters, it provides several design choices to develop an integration-oriented algorithm. Furthermore, we develop the  $8 \times 8$  post-loop filter with macroblock-based instead of frame-based filtering structure. There is no need to buffer the whole frame in advance. As a result, the modified post-loop filter achieves area-efficiency and is easily to be integrated into in-loop de-blocking filter. In the following, we detail the design method to support both standard-compliant in-loop and informative post-loop filters.

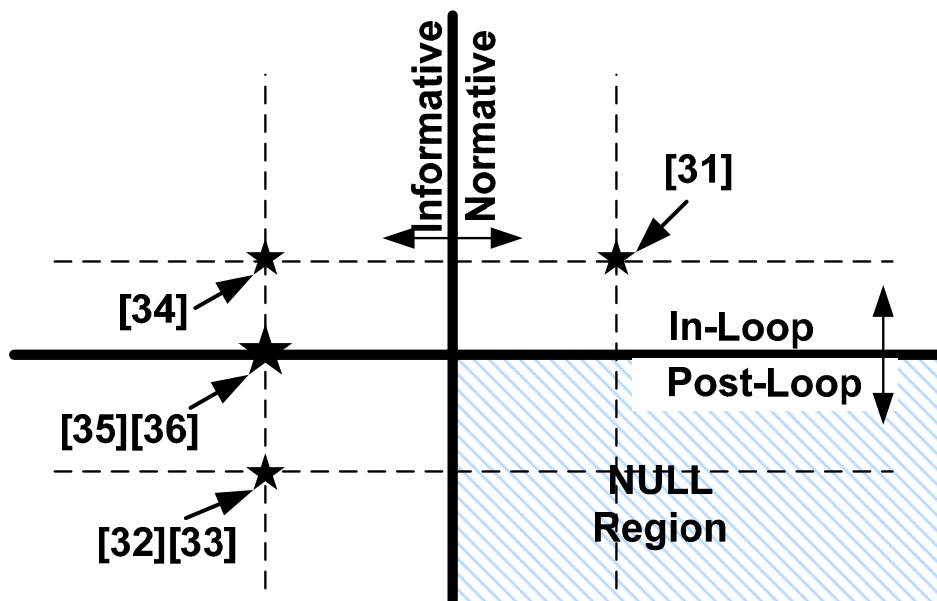


Figure 2.8: Various types of de-blocking filters

## 2.8.2 Algorithmic Preview

Due to a great diversity of deblocking filters in different standards, we tabulate each feature in Table 2.3. The filtering control decides the filtering order and the size of filtered boundaries. In general, most deblocking filters obey an order that executes on the horizontal edges first followed by the vertical edges. But, this order is different from that defined in H.264/AVC. As for the filtered boundary, the in-loop filter of H.264/AVC is applied to each boundary of  $4 \times 4$  sub-block while the post-loop filter is executed on that of  $8 \times 8$  block. With regard to the in-loop de-blocking filter in VC-1 [37], it is performed on the block boundaries of  $4 \times 4$ ,  $4 \times 8$ ,  $8 \times 4$ , and  $8 \times 8$ . However, no matter how they locate, those boundaries are easily accomplished since the  $4 \times 4$  sub-block boundary is the smallest edge and all boundaries can be considered as a super-set of  $4 \times 4$  sub-block.

Table 2.3: Features of deblocking filter in different standards.

Deblocking Filter		In-Loop Filter		Post-Loop Filter	
Standardization		Normative		Informative	
STANDARD		H.264/ AVC	WMV-9/ VC-1	MPEG-4 (Annex F.3)	H.263 (Annex J)
Filtering Control	Filtering Order	Vertical First	Horizontal First	Horizontal First	Horizontal First
	Filtered Boundary	$4 \times 4$	$4 \times 4 / 4 \times 8 / 8 \times 4 / 8 \times 8$	$8 \times 8$	$8 \times 8$
Filtering	Strength/Mode	5/2	2/1	2/2	12/1
Process	No. of input pixels	8	8	10	4

Filtering processes can be divided into three main parts. The first part of processes is the strength decision. It governs the filtering intensity in that edge. H.264/AVC employs a

boundary strength (i.e. bS spreads from 0 to 4, 5-strength) to calculate the strength in each filtering mode. VC-1 adopts the edge\_strength (i.e. only true or false, 2-strength) to realize the strength decision [37]. Moreover, MPEG-4 and H.263 feature 2-strength (i.e. eq\_cnt $\geq$ 6 or eq\_cnt $<$ 6) and 12-strength (i.e. strength=1~12) decisions respectively. The second part of processes is the mode decision which is comprised of strong and weak modes. For instance, in MPEG-4 Annex F.3 [38], Kim *et al.* [32] exploited smooth regions and default modes as strong and weak modes respectively. In H.264/AVC, List *et al.* [31] applied strong and weak modes when the boundary strength (i.e. bS) is equal to or less than 4 respectively. Excluding two mode decisions, there is one mode decision in VC-1 [37] and H.263 Annex J [39]. A third part of filtering processes is the edge filter. It operates on a specified edge, and smooth out the discontinuities with pre-defined coefficients. In general, the numbers of input pixels are related to the filtering performance as well as computational complexity. The in-loop filter takes 4-pixel on either side of the edge. The post-loop filter in MPEG-4 and H.263 requires 5-pixel and 2-pixel on either side respectively. However, only partial pixels will be modified. Figure 2.9 depicts the pixels that are involved in a filtering operation and the modified pixels for output. After previewing aforementioned features, we conclude that there are great diversities in those filters. Hence, a combined in/post-loop filter algorithm is of great challenge for saving design cost.

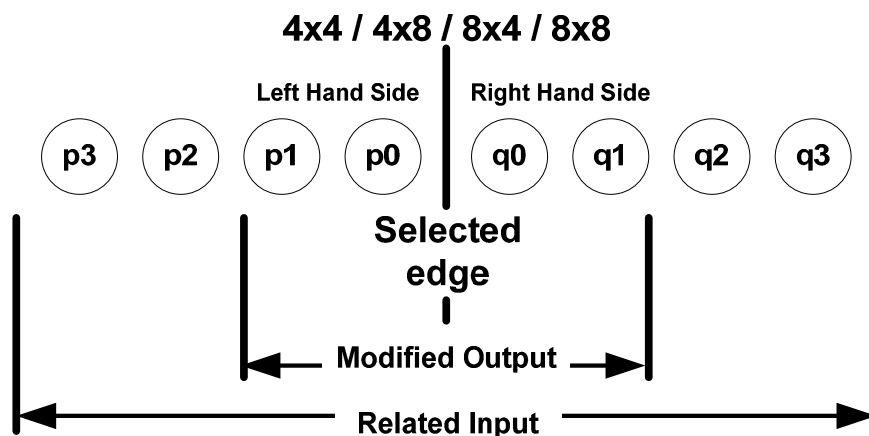


Figure 2.9: The modified and related pixels in the process of edge filter.

### 2.8.3 In/Post-loop Algorithm

Using a single algorithm to realize in-loop or post-loop filter is inferior since the source of blocking artifacts comes from a distinct quantization process, IDCT kernel and the motion compensated algorithm between MPEG-2 and H.264/AVC. From the experimental results, the quality improvement is very mild (only 0.04dB) when we replace a post-loop filter with an in-loop filter. To cope with this problem, we propose an integration-oriented algorithm which tightly combines H.264/AVC in-loop filter with MPEG-4 post-loop filter. Note that we choose the filter defined by MPEG-4 Annex F.3 as a MPEG-2 post-loop filter. Specifically, we keep the filtered boundaries of  $4\times 4$  and  $8\times 8$  in the in-loop and post-loop filters respectively. Additionally, to unify into a single architecture, the filtering order in post-loop filters has been changed from horizontal to vertical edges first. With regard to filtering processes, a triple-mode decision and triple pixel-in-pixel-out edge filter are proposed in the following sub-section so as to improve the hardware utilization. Moreover, they provide an easy exchange of different filter types without greatly changing a hardware prototype.

#### 2.8.3.1 Triple-mode decision

A triple-mode decision adopts a SKIP mode and resource sharing technique to reduce filtering complexity and integration cost respectively. Firstly, this decision has been applied to H.264/AVC and employs strong, weak and SKIP modes according to the boundary strength (bS). As to the post-loop filter in MPEG-4, Kim *et al.* [32] exploited the threshold  $T_2$  as 6 to distinguish between default (i.e. weak) and DC offset (i.e. strong) modes. However, it is very time-consuming because there is no skip conditions applied and all  $8\times 8$  block boundaries execute filtering processes. To alleviate this problem, we introduce

another threshold  $T_3$  to reduce the computation in Figure 2.10. Moreover, since fixed thresholds  $\{T_2, T_3\}$  cannot achieve better performance, we use the header information (e.g. MVD, CBP, MB\_TYPE) to adjust the thresholds dynamically. In Table 2.4, we propose a compound decision method to share the hardware resource since MPEG-4's  $\{T_2, T_3\}$  resemble H.264/AVC's  $\{bS, \alpha, \beta, t_{c0}\}$ . Moreover, we found that different bit rates contribute to the difference of the threshold  $T_2$ . Introducing a term of  $t_{T_2}$  as a function of QP makes it more robust in terms of the bit rate variations. In conclusion, the proposal reduces not only the computation through the SKIP mode but also the integration cost by the compound method.

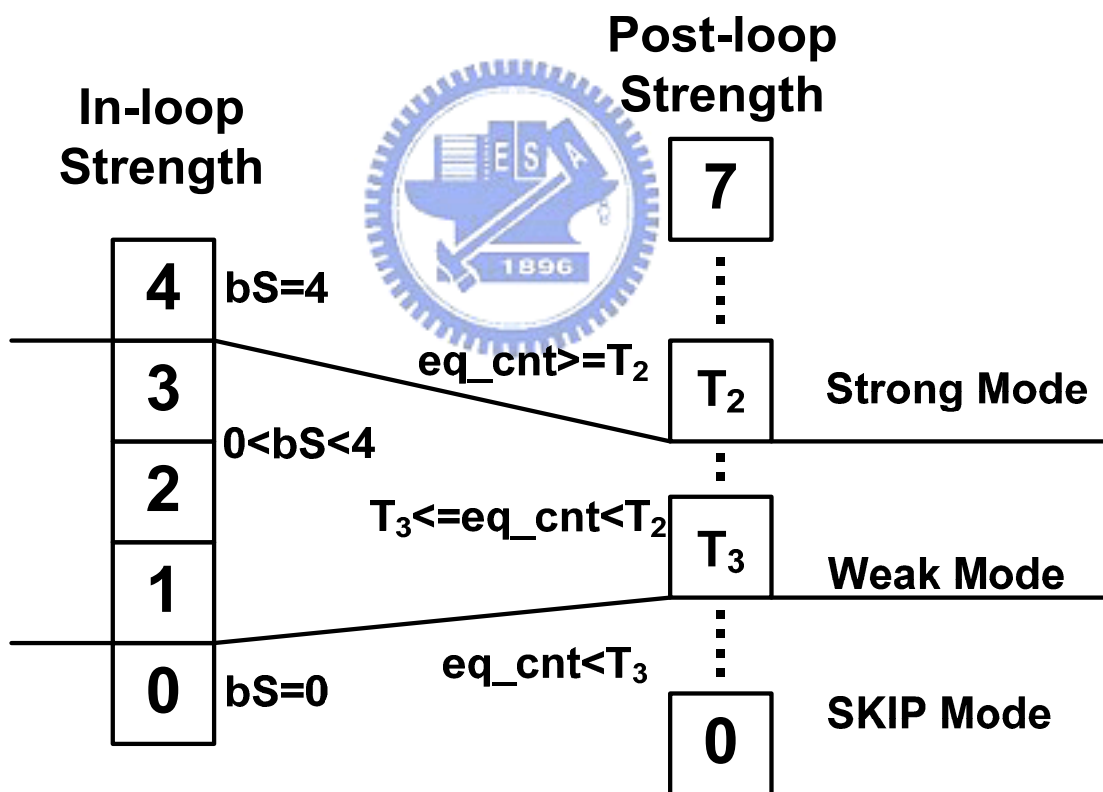


Figure 2.10: A triple-mode decision of the in/post-loop filter.

Table 2.4: A compound method for the strength decision.

In/Post-loop Decision Method		In-loop	Post-loop	
Block modes and conditions	Semantics	bS	T <sub>3</sub>	T <sub>2</sub>
One of the blocks is intra and the edge is a macroblock edge	MB_TYPE==INTRA && MB Edge	4	T <sub>3i</sub> *	T <sub>2i</sub> *
One of the blocks is intra	MB_TYPE==INTRA	3	T <sub>3i</sub>	T <sub>2i</sub>
One of the blocks has coded residuals	CBP != 0	2	T <sub>3i</sub> +1	T <sub>2i</sub> +1
Difference of block motion $\geq 4$ at luma sample difference	MVD <sub>x</sub> $\geq 4$    MVD <sub>y</sub> $\geq 4$	1	T <sub>3i</sub> +2	T <sub>2i</sub> +1
Else	Else	0	T <sub>3i</sub> +2	T <sub>2i</sub> +1+t <sub>T2</sub> (QP)

T<sub>3i</sub>\*, T<sub>2i</sub>\*: the initial value of T<sub>3</sub> and T<sub>2</sub>.

### 2.8.3.2 Triple pixel-in-pixel-out edge filter

We develop a triple P-i-P-o edge filter to reduce the integration cost. In the post-loop mode, the edge filter retains default mode and discards the DC offset mode because the default mode is of the prime concern while the DC offset mode is broadly similar to the strong mode of the in-loop filter. That is, we can replace the edge filter of DC offset mode with that of “bS=4” (strong mode) for an integration-oriented design approach. We change the approximated DCT kernel (i.e. [2 -5 5 -2]) to [2 -4 4 -2]. As a result, we make use of shifters instead of constant multipliers. Moreover, to merge the edge filter in the weak mode, we modify the numbers of input pixels to 8 pixels in the post-loop filter. Thus, the numbers of input pixels in the in-loop and post-loop filters are equivalent. Specifically, Figure 2.11 depicts the detailed circuit of the weak filter. Generally, it needs a great number of operations and greatly influences the visual quality. It takes 4-pixel (i.e. p0~p3, q0~q3) on

either side of the boundary to realize interpolation procedures. In particular, a pixel-wise difference is applied, and a delta metric is generated. A CLIP<sup>8</sup> operation limits the delta metric between  $y$  (i.e. Upper Bound) and  $x$  (i.e. Lower Bound). Finally, the CLIP's outputs add and subtract the raw pixels to obtain the filtered results. Because the weak filter is the most quality-intensive process in the deblocking filter, we share most of operations except "Delta Generation" and make a better trade-off between visual quality and area efficiency. As a result, the synthesized logic gate counts can be reduced by 30% compared to the preliminary design that implements in-loop or post-loop filter separately. In conclusion, three data flows (i.e. strong, weak and SKIP) and related pseudo codes are highlighted in Figure 2.12, and some modifications are made on the post-loop filter to improve the integration efficiency. These modifications definitely reduce the integration overhead with a penalty of slight performance loss and will be addressed in the next sub-section.

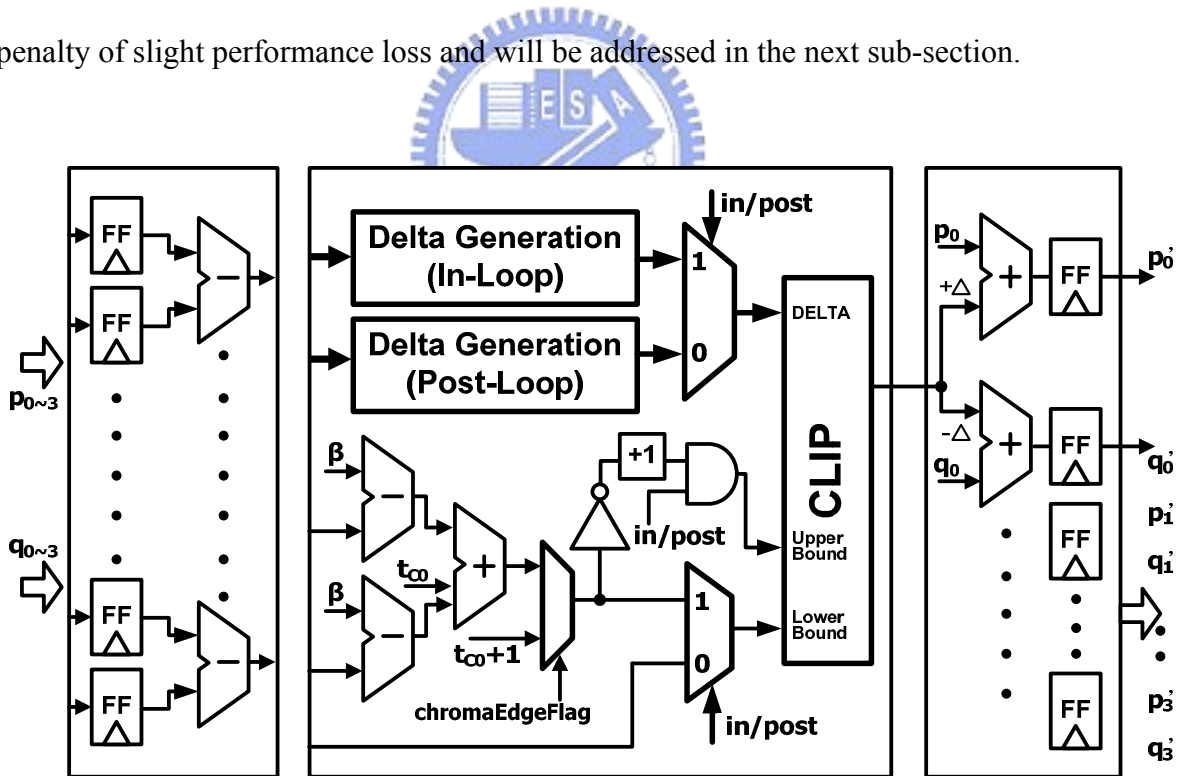


Figure 2.11: Triple P-i-P-o edge filter in weak mode.

<sup>8</sup> 
$$\text{CLIP}(x,y,z) = \begin{cases} x & ; z < x \\ y & ; z > y \\ z & ; \text{otherwise} \end{cases}$$

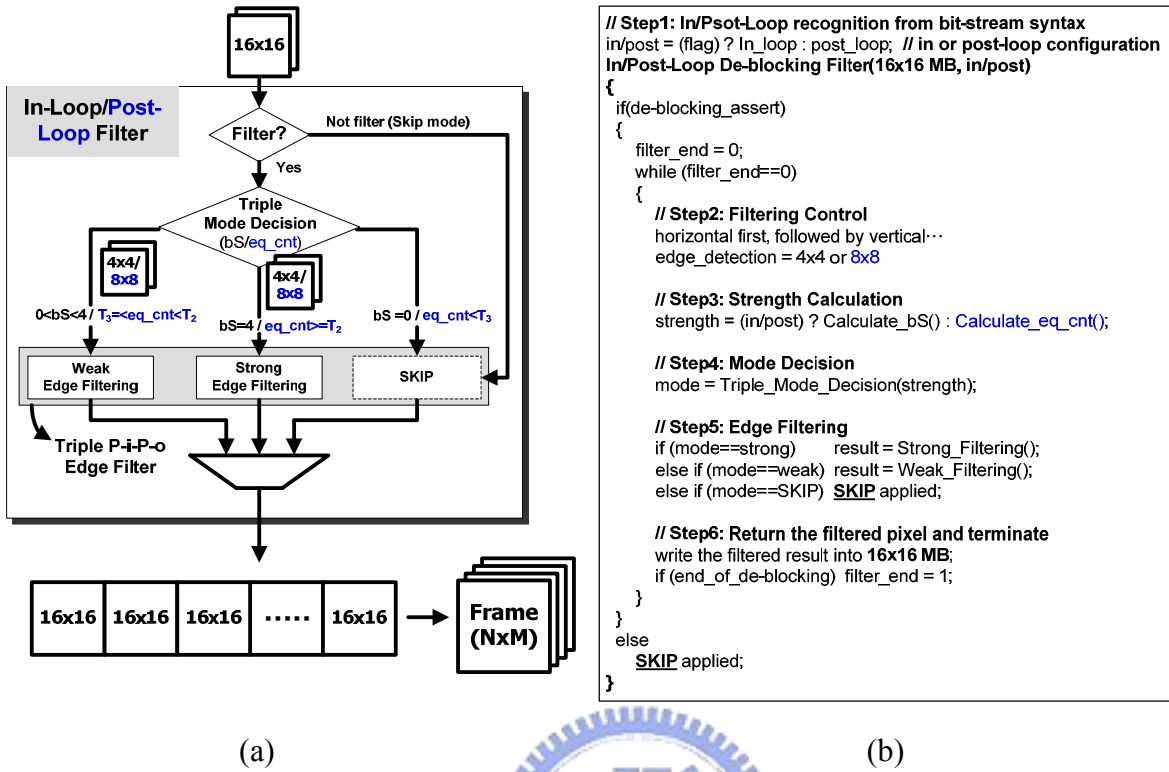


Figure 2.12: The (a) data flow and (b) pseudo code of the in/post-loop algorithm.

## 2.8.4 Performance Evaluation

The modifications of the post-loop filter improve the integration efficiency at a cost of slight performance degradation. For the experiments of MPEG-4's post-loop filter, the thresholds of  $T_{2i}=5$  and  $T_{3i}=0$  (see Table 2.4) are employed without loss of generality. Furthermore, we adopt Table 2.5 as the induced term of  $t_{T2}$ . QP stands for “quantizer precision”, and we use 5-bit as a default value that ranges from 0 to 31. All alterations of the MPEG-4's post-loop algorithm have been addressed, and specific results are given in Table 2.6. All sequences are defined in CIF (352×288) and INTRA PERIOD 15 with 30fps throughout 300 frames. We show that the performance degradation is less than 0.05dB as compared to the MPEG-4's post-loop filter [38]. From the subjective point of view, we capture the 20<sup>th</sup> frame to give a comparison in Figure 2.13.



Table 2.5: Values of  $t_{T2}$  with a function of QP.

QP	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$t_{T2}$	2	2	1	1	1	1	1	1	0	0	0	0	0	0	0	0

Table 2.5(continued): Values of  $t_{T2}$  with a function of QP.

QP	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$t_{T2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.6: The post-loop filtering performance in terms of luma PSNR.

MPEG-4 Decoder		PSNR-Y [dB]		
Bit Rate	CIF Sequence	w/o filter	MPEG-4 Annex F.3 filter	Proposed
150kbps	Table	32.01519	32.13722	32.13138
	Mobile calendar	24.97704	25.00832	25.04150
	Mother & daughter	38.73812	39.00201	39.02740
	Stefan	27.02727	27.14439	27.10813
450kbps	Table	37.11400	37.18894	37.19088
	Mobile calendar	27.43971	27.49859	27.48667
	Mother & daughter	42.85162	42.99503	42.99579
	Stefan	30.63070	30.77906	30.72632
1500kbps	Table	42.83698	42.84450	42.87695
	Mobile calendar	34.50489	34.57378	34.56172
	Mother & daughter	46.33545	46.48835	46.44548
	Stefan	38.36032	38.48165	38.44038



(a) w/o filter.



(b) Original filter in MPEG-4.



(c) Proposed.

Figure 2.13: The subjective quality comparison.

## 2.9 Summary

Figure 2.14 summarizes our work on combined MPEG-2/H.264 video decoder. First, a register-sharing technique efficiently reduces the required storage. Second, a table-merging method combines the identical codeword and prefix among tables for saving table size. Third, although area on MPEG-2 DC prediction is far less than H.264/AVC intra prediction, combined intra/inter prediction is a good point for further area reduction. Fourth, IDCT is the most diverse algorithm over the whole modules. A recursive algorithm and multiple constant multiplication techniques are presented to improve the hardware utilization in an algorithmic level. Fifth, interpolators require a great deal of adders and are the most area-critical module in the motion compensation design. To cope with this problem, a combined luma/chroma interpolator is proposed. Finally, a combined algorithm for simultaneously meeting in-loop and post-loop deblocking filters is presented to save area. To this end, we propose a triple-mode decision and triple P-i-P-o edge filter to realize the in/post-loop algorithm. Experimental results exhibits that 30% of area reduction is achieved without great loss of visual quality. Let's sum up aforementioned cost reduction. Figure 2.14 shows that 20% of area reduction can be achieved in a system point of view. Moreover, this low-cost design approach is applicable for multi-standard applications, such as HD-DVD and DVB-H systems.

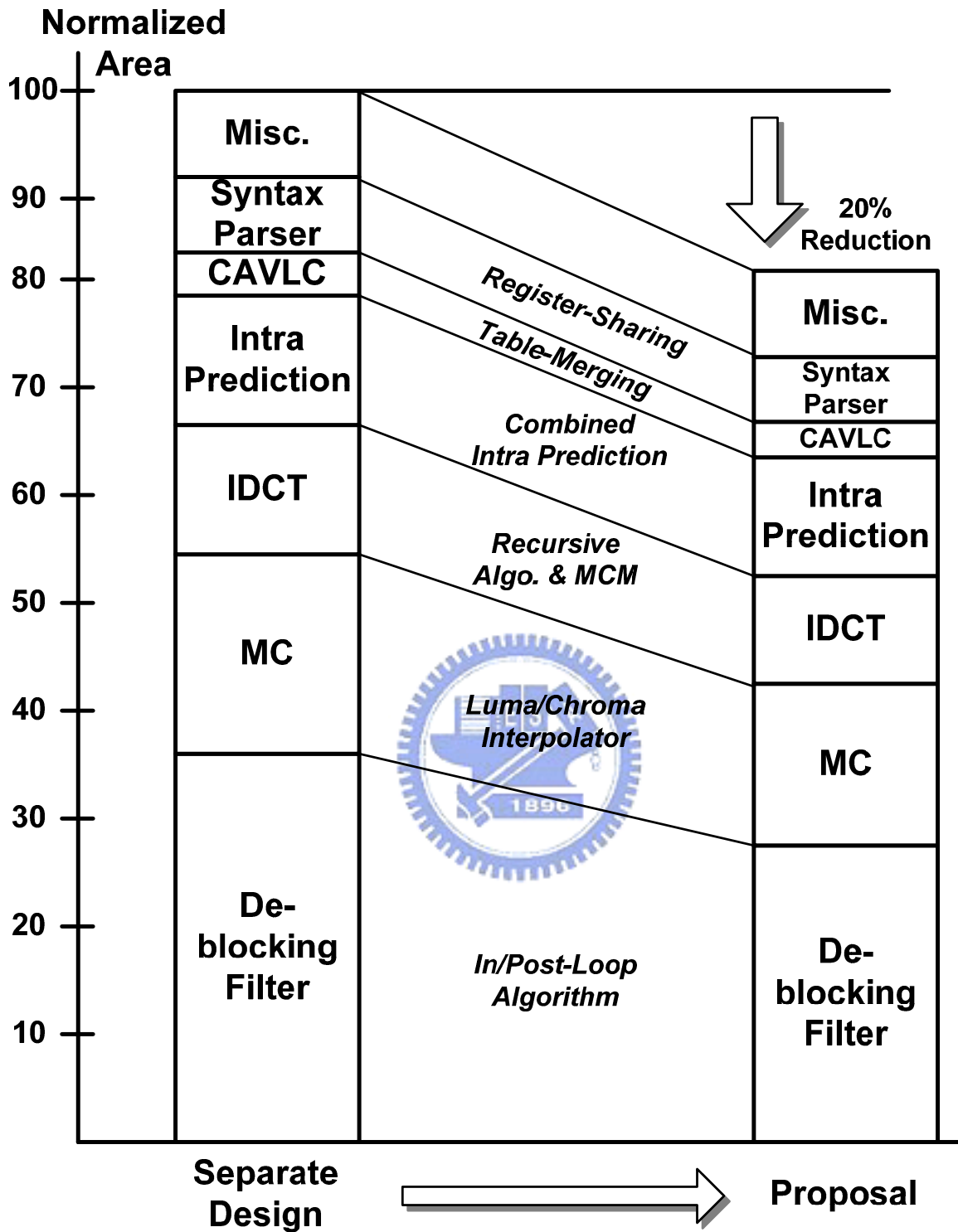


Figure 2.14: A cost summary of dual MPEG-2/H.264/AVC video decoder.