EFFICIENT IMPLEMENTATION OF A SET OF LIFTING BASED WAVELET FILTERS

Kishore Andra[*], Chaitali Chakrabarti [*], Tinku Acharya[*,+]
[*] Arizona State University, Tempe, Arizona, 85287, USA.
[+] Intel Corporation, Chandler, Arizona, USA.

ABSTRACT

Lifting based wavelet transform implementation not only helps in reducing the number of computations but also achieves lossy to lossless performance with finite precision. In this paper we first do a precision analysis for the set of seven filters proposed by the JPEG2000 verification model. We determine the precision required to implement the filters using fixed point 2's complement arithmetic for lossless as well as lossy coding. Next we propose a unified architecture for implementing this set of filters for both the forward and the inverse transform.

1. INTRODUCTION

Wavelet based image codecs are being used extensively because of their capabilities to support transmission by progressive quality and resolution, region of interest coding, ease of compressed image manipulation etc. The conventional convolution based implementation of the Discrete Wavelet Transform (DWT) has high computational and memory requirements. Recently, the lifting based implementation of DWT [1,2] has been proposed to overcome these drawbacks. In addition, it supports in place computation, symmetric inverse transform and even lossless performance with finite precision.

The main feature of the lifting based DWT scheme is to break up the high pass and low pass filters into a sequence of upper and lower triangular matrices, and thus convert the filter implementation into banded matrix multiplications. Such a scheme makes it possible to achieve an integer-to-integer transform [4,5] where, if the input data is in integer format then data can be maintained in the same format through out the transform by introducing a rounding function. Due to this property, the transform is reversible (i.e. lossless) and is called the Integer Wavelet Transform (IWT). It should be noted that filter coefficients need not be integers for IWT.

In this paper, we consider the lifting based implementation of the filters proposed in the JPEG 2000 Verification Model [3]: (5,3), C(13,7), S(13,7), (2,6), (2,10), (6,10) and (9,7). We determine the precision required for signals and the filter coefficients to implement the above filters using the Single Sample Overlap Wavelet Transform (SSOWT) in fixed point, 2's complement arithmetic. We find that a 14 bit precision is required for a satisfactory lossy performance and 16 bit precision for lossless performance. Next, we propose an architecture for these filters that produces an output every cycle for the JPEG2000 default filters [(5,3) for lossless mode and (9,7) for lossy mode]. The architecture is capable of carrying out DWT/IDWT in row-column fashion. It consists of two row processors, two column processors and two memory modules.

The rest of the paper is organized as follows. In section 2 we give a brief introduction to lifting based DWT. In section 3 we determine the precision that is required for fixed point implementation. In section 4 we briefly describe the proposed architecture.

2. LIFTING SCHEME - PRELIMINARIES

The basic principle of the lifting scheme [1,2] is to factorize the polyphase matrix of a wavelet filter into a sequence of alternating upper and lower triangular matrices and a diagonal matrix with constants. The factorization is obtained by using an extension of the Euclidean algorithm. The resulting formulation can be implemented by means of banded matrix multiplications.

Let $\tilde{h}(z)$ and $\tilde{g}(z)$ be the low and high pass analysis filters and $h(z)$ and $g(z)$ be the low and high pass synthesis filters. The filters can be divided into even and odd parts as

$$\tilde{h}(z) = \tilde{h}_e(z^2) + z^{-1}\tilde{h}_o(z^2) \qquad h(z) = h_e(z^2) + z^{-1}h_o(z^2)$$
$$\tilde{g}(z) = \tilde{g}_e(z^2) + z^{-1}\tilde{g}_o(z^2) \qquad g(z) = g_e(z^2) + z^{-1}g_o(z^2)$$

The polyphase matrices are then defined as -

$$\tilde{P}(z) = \begin{bmatrix} \tilde{h}_e(z) & \tilde{h}_o(z) \\ \tilde{g}_e(z) & \tilde{g}_o(z) \end{bmatrix} \qquad P(z) = \begin{bmatrix} h_e(z) & g_e(z) \\ h_o(z) & g_o(z) \end{bmatrix}$$

It has been shown in [2] that if $(\tilde{h}, \tilde{g})$ is a complementary filter pair, then $\tilde{P}(z)$ can always be factored into lifting steps as shown below : (K_1 and K_2 are constants)

$$\tilde{P}_1(z) = \begin{bmatrix} K_1 & 0 \\ 0 & K_2 \end{bmatrix} \prod_{i=1}^{m} \begin{bmatrix} 1 & \tilde{s}_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \tilde{t}_i(z) & 1 \end{bmatrix} \quad or$$

$$\tilde{P}_2(z) = \begin{bmatrix} K_1 & 0 \\ 0 & K_2 \end{bmatrix} \prod_{i=1}^{m} \begin{bmatrix} 1 & 0 \\ \tilde{t}_i(z) & 1 \end{bmatrix} \begin{bmatrix} 1 & \tilde{s}_i(z) \\ 0 & 1 \end{bmatrix}$$

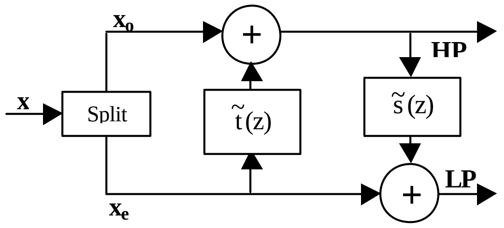

Fig.1 Lifting Scheme based on $\tilde{P}_1(z)$ factorization

Scheme 1 corresponds to the $\tilde{P}_1(z)$ factorization, and is shown in Fig.1. Here the low-pass samples (even terms) are multiplied by the time domain equivalent of $\tilde{t}(z)$, and are added to the high pass samples (odd terms) in the first step. In the second step, the updated high pass samples are multiplied by the time domain equivalent of $\tilde{s}(z)$ and are added to the low-pass samples. In Scheme 2 which corresponds to the $\tilde{P}_2(z)$ factorization, the low-pass terms are calculated in the first step ($\tilde{s}(z)$ is used) and the high pass terms are calculated in the second step ($\tilde{t}(z)$ is used). For IWT, in each lifting step, the result of the filtering operation has to be rounded right before addition or subtraction.

In both the schemes, if a diagonal matrix is present in the factorization, the low pass coefficients are multiplied with K_1 and the high pass coefficients are multiplied with K_2. In such cases, the IWT cannot be achieved just by rounding. Then IWT can be achieved by splitting the diagonal matrix into extra lifting steps [4] and then applying rounding. We do not explore this option in this paper.

The wavelet filters are classified into (i) Case 1 filters which consists of (5,3), C(13,7), S(13,7) and can be factorized into two matrices; (2,6) and (2,10) which can be factorized into three matrices (ii) Case 2 filters which consists of (9,7) and (6,10) and can be factorized into four matrices and a diagonal matrix.

3. PRECISION ANALYSIS

We have carried out a comparison study between the floating point and fixed point implementations to determine the number of bits required in a fixed point implementation for satisfactory lossy performance and also for lossless performance. We have used three gray scale images, Baboon, Barbara and Fish, each of size 513x513 and carried out the study for 5 levels of decomposition. We have verified the results from the study with 15 gray scale images from the USC-SIPI image database.

3.1 Lifting Coefficients

The magnitude of the lifting coefficients for the filters we have considered range from 0.046875 to 1.58613. In order to convert all the filter coefficients to integers, the coefficients are multiplied by 256 (i.e. shifted left by 8 bits). The range of coefficients is now 12 to 406. This implies that we require 10 bits (2's complement representation) for the lifting coefficients. At the end of each multiplication, the product is shifted right by 8 to get the required result. This is implemented in hardware by rounding the 8 least significant bits.

3.2 Signal values

The signal values have to be shifted left too in order to increase the precision; the extent of the shift is determined using image quality analysis. In this work we have experimented with a shift ranging from 0 to 5 bits (we call them the Additional Bits, ABs). It should be noted that, filters (5,3) and (2,6) can be implemented with just shifts as their lifting coefficients are multiples of 2. But we have carried out the analysis by multiplications for comparison purposes. If the filters are implemented with shifts, additional bits are not required.

In conventional fixed point implementation, instead of shifting the input samples, the coefficients are shifted by required number of extra bits. Consider the general structure of a lifting based filter implementation with SSOWT -

$$y = a\,(x1 + x2) + b\,(x3 + x4) + x5$$

where a *and* b are the filter coefficients, *x's* are the signal samples and *y* is the transform value. If the filter coefficients alone are shifted, an extra shifting operation has to be performed on the *x5* term to maintain the data alignment.

3.3 Rounding

We have considered a method for rounding wherein, both +ve and -ve numbers are rounded towards $+\infty$ i.e. the numbers would be rounded to next higher integer (for ex. 965.50 $\rightarrow$ 966 and -965.50 $\rightarrow$ -965). To achieve this, a '1' should be added for both, +ve and –ve numbers whenever the MSB of the bits being truncated is a '1'. If MSB = '0' the number can be just truncated. It should be noted that instead of applying rounding on the result of the filter operation (which results in bigger accumulators) as in [4], it is applied to the individual terms of the filter result.

Example : Consider the general lifting structure. Let a = 0.2345, b = 1.4567, x1 = 24, x2 = 43, x3 = 156, x4 = 56 and x5 = 10. The floating point implementation result is y = 334.5319. Let us assume that coefficients are shifted left by 8 bits (and rounded to nearest integer) and number of *ABs* = 2. Then a = 60, b = 373, x1 = 96, x2 = 172, x3 = 624, x4 = 224 and x5 = 40. The products are 60(96+172) = 16080 and 373(624+224) = 316304. Shifting the product right by 8 bits and rounding will yield 63 and 1236. So y = 63 + 1236 + 40 = 1339. This should be interpreted as round[(1337>>2)+two bits]=round[334.75]=335.

3.4 Results

All through this work we define *SNR* as

$$SNR\ (dB) \;=20\ \log_{10}\left(\frac{\sum\left|Signal\right|}{\sum\left|Signal-\ fixed\ \ point\ implementation\right|}\right)$$

For forward transform, *Signal* corresponds to the values obtained by the floating point implementation. For forward transform followed by the inverse transform, *Signal* corresponds to the original image data.

The SNR values for the Baboon image after 5 levels of forward transform with the product terms formed by truncating and rounding the 8 LSB's are given in Table 1. We see that the SNR for the forward transform is not degraded by truncation.

The SNR values for the Baboon image after 5 levels of forward and inverse transform with truncation and rounding are given in Table 2. From Table 2, we see that except for (5,3) and (2,6) filters and to some extent (2,10) filter, truncation severely degrades the performance of the filters. But if (5,3) and (2,6) filters have to be implemented with multiplications, then truncation will be sufficient to achieve the lossless performance.

From Table 1 it is clear that more the number of ABs better the performance of the fixed point implementation. However, having more than 2 ABs has little effect on (6,10) and (9,7) filters. From Table 2 we see that with 2 ABs the SNR for all the filters is > 30 dB with rounding. So we can conclude from Table 1 and Table 2, that 2 ABs are enough for satisfactory lossy performance. Further from Table 2, we see that with 5 ABs all the filters except for (6,10) and (9,7) filters are lossless with rounding. So we conclude that 2 ABs are needed for a satisfactory lossy performance and 5 ABs for lossless performance. A similar trend was observed for the other two images.

Once the number of ABs are fixed, we need to determine the width of the data path. This can be done by observing the maximum (minimum) values for the transformed values at the

ABs	(5,3)		C(13,7)		S(13,7)		(2,6)		(2,10)		(6,10)		(9,7)	
	Trunc.	Round	Trunc.	Round	Trunc.	Round	Trunc.	Round	Trunc.	Round	Trunc.	Round	Trunc.	Round
0	28.458	29.788	19.801	27.870	21.678	28.120	28.478	30.654	22.725	29.349	18.154	24.858	18.225	24.861
1	36.190	37.536	25.866	33.758	27.800	34.116	34.991	37.553	28.759	36.011	23.989	30.364	24.219	28.458
2	44.380	45.687	32.048	39.590	34.046	39.962	41.945	43.677	34.861	42.324	29.502	35.068	28.759	30.376
3	52.357	52.974	38.449	45.264	40.520	45.843	50.631	52.368	41.060	48.391	34.257	38.140	30.864	31.103
4	63.513	64.674	45.379	53.146	47.603	53.616	61.063	63.508	47.425	54.234	37.525	39.560	31.478	31.358
5	69.761	70.924	51.587	59.133	53.751	59.630	70.097	71.087	54.124	59.531	39.193	40.042	31.574	31.430

Table 1 SNR values after forward transform for the Baboon image

ABs	(5,3)		C(13,7)		S(13,7)		(2,6)		(2,10)		(6,10)		(9,7)	
	Trunc.	Round	Trunc.	Round	Trunc.	Round	Trunc.	Round	Trunc.	Round	Trunc.	Round	Trunc.	Round
0	12.985	18.949	3.990	27.399	1.783	27.090	18.800	18.977	18.157	18.983	5.069	28.859	-6.602	24.561
1	19.614	24.405	10.159	31.042	7.944	30.857	26.338	24.525	25.114	24.565	11.322	32.727	-0.571	30.527
2	25.811	30.208	16.176	37.478	13.928	36.480	32.680	31.245	31.627	31.325	17.457	36.164	5.449	36.374
3	32.797	34.507	22.244	48.720	20.016	48.213	39.825	36.821	38.793	36.803	23.806	37.876	11.495	43.620
4	36.964	77.467	28.013	113.224	26.322	110.302	70.646	73.633	48.550	70.892	30.333	38.883	17.490	56.711
5	inf	inf	33.094	inf	32.896	inf	inf	inf	99.819	inf	36.070	39.312	23.479	77.581

Table 2 SNR values after forward and inverse transform for the Baboon image

end of each level of decomposition and taking the largest (smallest) among them. The maximum and minimum values with ABs = 2 and 5 are given in Table 3 and Table 4.

From Table 3 and Table 4 we see that the (9,7) filter generates both maximum and minimum values in case of Baboon and Barbara images, while the (6,10) filter generates maximum and minimum values for the Fish image. Based on these results, we need 13 bits to represent the transform values with 2 ABs. However in some stray cases, the internal precision is of 14 bits magnitude. So the width of the data path required for lossy transform is 14 bits. Similarly we can deduce, from Table 4, that a 16 bit wide data path is required for lossless performance. All the above results have been verified with USC-SIPI database images - 5.2.08-10, 7.1.01-04, 7.1.06-10, boat, elaine, ruler and gray21 from the Miscellaneous directory.

Filter	Baboon		Barbara		Fish	
	Max	Min	Max	Min	Max	Min
(5,3)	1080	-951	746	-662	609	-654
C(13,7)	781	-772	742	-629	519	-512
S(13,7)	785	-744	723	-657	486	-514
(2,6)	904	-1062	783	-729	571	-645
(2,10)	1153	-1377	930	-1087	706	-737
(6,10)	1107	-760	1273	-903	1025	-1403
(9,7)	3116	-2421	2390	-2442	864	-1127

Table 3 Maximum and minimum values for ABs = 2

Filter	Baboon		Barbara		Fish	
	Max	Min	Max	Min	Max	Min
(5,3)	8642	-7611	5950	-5296	4876	-5253
C(13,7)	6250	-6183	5930	-5036	4147	-4096
S(13,7)	6287	-5951	5778	-5261	3887	-4097
(2,6)	7216	-8487	6269	-5837	4556	-5171
(2,10)	9226	-11008	7429	-8705	5664	-5891
(6,10)	8877	-6077	10192	-7231	8175	-11248
(9,7)	24920	-19370	19073	-19524	6871	-8991

Table 4 Maximum and minimum values for ABs = 5

4. PROPOSED ARCHITECTURE

The proposed architecture calculates DWT/IDWT in the row-column fashion on a block of data of size NxN. It is an extension of the DWT architecture proposed in [6]. The architecture consists of a Row module (two Row Processors RP1 and RP2), a column module (two Column Processors CP1,CP2) and two memory modules (MEM1, MEM2). To perform DWT(IDWT), data has to be written into MEM1 along the rows(columns). In the rest of the paper, we assume that data is stored in transposed form for IDWT and discuss all the details in terms of rows for both DWT and IDWT.

To perform the DWT, the architecture reads in the block of data, carries out the transform, and outputs the LH, HL, HH data at each level of decomposition. The LL data is used for the next level of decomposition. To perform the IDWT, all the sub bands from the lowest level are read in. At the end of the inverse transform, the LL values of the next higher level are obtained. The transform values of the three bands (LH,HL and HH) are then read in and the IDWT is carried out on the new data set.

For the Case 1 filters (i.e. when lifting is implemented by two /three factorization matrices), processors RP1 and RP2 read the data from MEM1 perform the DWT/IDWT along the rows, and write the data into MEM2. To reduce the latency and the memory requirements, the column processors calculate the column wise transforms along the *rows*. Processor CP1 reads the data from MEM2, performs the column wise DWT/IDWT along alternate *rows*, and writes the data into MEM2 and MEM1/Ext.MEM. Processor CP2 also reads the data from MEM2 and writes to MEM1/extermanl memory, and performs the column wise DWT/IDWT along the *rows* that CP1 did not work on. Fig. 2 describes the data flow for the Case 1 filters.

For the Case 2 filters (i.e. when lifting is implemented by four factorization matrices), there are two passes. In each of the passes, RP1 and RP2 read in the data, execute the first two matrix multiplications and write the result into MEM2. CP1 and CP2 execute the next two matrix multiplications. In the first pass, both CP1 and CP2 write the results into MEM1 in column major fashion and in the second pass, CP2 writes the LL subband data to MEM1, while other three subbands are written to Ext.MEM.

4.1 Row and Column processor design

Each filter requires a different configuration of adders, multipliers and shifters in the data path in order to generate an output every cycle. We have considered a configuration that generates an output every clock cycle for the default JPEG2000 filters (5,3) (lossless) and (9,7) (lossy) filters. The proposed architecture consists of four processors, where each processor consists of 2 adders, 1 multiplier and 1 shifter.

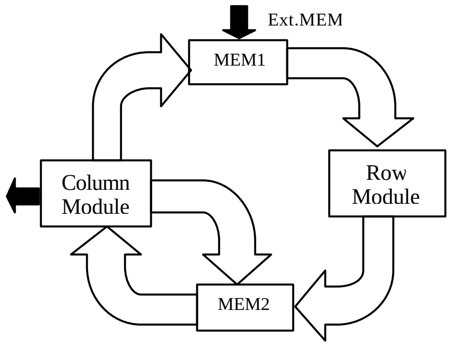

Fig. 2 Data flow for Case 1 filters

For lossy (lossless) coding, adders and shifters have to handle 14 (16) bits data, while multipliers have to multiply 14 (16) bit data with a 10 bit coefficient.

4.2 Memory

The proposed architecture consists of two memory modules, (MEM1 and MEM2). Data in MEM1 is written by Ext.MEM, CP1 and CP2, and is read by RP1 and RP2. The data in MEM2 is written by RP1, RP2 and CP1, and read by CP1 and CP2. Each module consists of 4 banks with a maximum of 4 ports (read and write ports combined) per bank. The memory design and management is significantly different from that in [6].

MEM1 module **–** It consists of 4 banks ($MEM1_0$, $MEM1_1$, $MEM1_2$ and $MEM1_3$). For the Case 1 filters, for the forward transform, we need 2 banks ($MEM1_0$, $MEM1_1$), while for the inverse transform we need 3 banks ($MEM1_0$, $MEM1_1$, $MEM1_2$). For the Case 2 filters, we need all the 4 banks for both the forward and inverse transforms. Each bank contains either odd samples or even samples of a row or column. The memory banks in MEM1 module read in the whole block in the beginning during the forward transform, and read in the whole block at the last level during the inverse transform. As a result all the four memory banks are of size $N \times \lceil N/2 \rceil$.

Filter	$MEM2_0$	$MEM2_1$	$MEM2_2$	$MEM2_3$
(5,3)	1 row	1 row	1 row	1 row
C(13,7)	2 rows	2 rows	1 row	3 rows
S(13,7)	2 rows	2 rows	1 row	3 rows
(2,6)	1 row	1 row	1 row	2 rows
(2,10)	2 rows	2 rows	1 row	4 rows
(6,10)	2Ta+3Tm+5 *(elements)*		-	-
(9,7)	2Ta+3Tm+3 *(elements)*		-	-

Table 5 Size of MEM2 module banks

MEM2 Module **-** It also consists of 4 banks ($MEM2_0$, $MEM2_1$, $MEM2_2$ and $MEM2_3$). We require all the 4 banks for the Case 1 filters and each bank contains complete row(s)/column(s) of data. For the Case 2 filters, we require 2 banks ($MEM2_0$, $MEM2_1$).and each bank contains the odd or even samples. The total memory required for the filters in MEM2 banks is given in Table 5 (Ta is adder/shifter delay and Tm is multiplier delay, a multiple of Ta).

4.3 Control

Control signals are needed primarily to maintain the steady flow of data to and from the processors. Our design consists of local controllers in each of the processors which communicate with each other by with hand shaking signals. Each local controller consists of three components – Counter, Memory signal generation unit and Address generation unit.

4.4 Timing

The total time required, for one level of decomposition of an NxN block, for all the filters is given in Table 6. Here, Ta is the delay of the adder, Ts is the delay of the shifter, and Tm is the delay of the multiplier. Timing is based on the values obtained by hand scheduling.

Filter	Timing
(5,3)	$2\lfloor N/2 \rfloor + 2Ta + 2Ts + N + 3 + \lfloor N/2 \rfloor N$
(13,7)	$7N + 6Ta + 2Tm + 4 + \lfloor N/2 \rfloor 2N$
(2,6)	$3\lfloor N/2 \rfloor + 6Ta + Tm + 3 + \lfloor N/2 \rfloor N$
(2,10)	$5N + 7Ta + Tm + 3 + \lfloor N/2 \rfloor 2N$
(9,7)	$2(4Ta + 6Tm + 6 + N\lfloor N/2 \rfloor)$
(6,10)	$2(3Ta + 6Tm + 6 + 2N\lfloor N/2 \rfloor)$

Table 6 Time required for a NxN block (one level)

4.5 Implementation

The architecture has been implemented in behavioral VHDL and simulated using Mentor Graphics VHDL compiler and Model Sim simulator running on Solaris on Ultra Sparc 10 machine. The adder and shifter are assumed to have a one clock cycle delay, where as the multiplier has a four cycle delay and is pipelined to 4 levels. The VHDL simulation results match exactly with C code simulations. The code is available at - http://www.public.asu.edu/~kishorea/lifting.

REFERENCES

1. I. Daubechies and W. Sweldens, " Factoring wavelet transforms into lifting schemes", The J. of Fourier Analysis and Applications, Vol. 4, 247-269, 1998.
2. W. Sweldens, "The lifting scheme: A new philosophy in biorthogonal wavelet constructions", Proceedings of SPIE, 2569, 68-79, 1995.
3. JPEG2000 Verification Model 6.0.
4. A.R. Calderbank, I. Daubechies, W. Sweldens and B-L. Yeo, " Wavelet transforms that map integers to integers", Applied and Computational Harmonic Analysis, Vol. 5, 332-369, July, 1998.
5. M. D. Adams and F. Kossentini, "Reversible Integer-to-Integer Wavelet Transforms for Image Compression: Performance Evaluation and Analysis", IEEE Trans. on Image Processing, vol. 9, 1010-1024, Jun. 2000.
6. K. Andra, C. Chakrabarti and T. Acharya, "A VLSI architecture for lifting based wavelet transform", IEEE workshop on Signal Processing Systems (SiPS 2000), 70-79, Oct. 2000.