QUANTIZATION EFFECT ON VLSI IMPLEMENTATIONS FOR THE 9/7 DWT FILTERS.

Vassilis Spiliotopoulos, N. D. Zervas, Yiannis Andreopoulos, G. Anagnostopoulos, Costas E. Goutis
University of Patras, Dept. of ECE
VLSI Design Lab., Rio 26500, Greece
bspili@ee.upatras.gr

Abstract: In this paper, two basic approaches for implementing the 9/7 Filtering Unit, used in the Discrete Wavelet Transform, are addressed. The first is the lifting scheme approach and the second is the conventional, convolutional filter approach. Two architectures are examined for each approach, a simple – straightforward one and an optimized one, substituting the multipliers used for scaling with shift – add operations. The quantization of the constants used in the calculations is thoroughly explored and the selection of the data-path bit-width is addressed. Experimental results based on hardware implementation, for several quantizations and for the different hardware architectures of the 9/7 filtering units are given.

I. INTRODUCTION

Many image and video compression techniques are based on the Discrete Wavelet Transform (DWT). A great number of analysis / synthesis filters for the DWT have been proposed in the past. However, Antonini's 9/7 filter [1] is the most popular one, since it combines good performance and rational filter length. It is stressed here that the 9/7 filter is a default filter of the upcoming JPEG2000 [2] standards and included in the MPEG4 [3] standard.

In this paper, aspects related to filter coefficients' quantization and hardware architecture of the filtering unit are exhaustively explored. Specifically, Section II focuses on lifting scheme-based implementation while Section III examines the conventional convolutional approach. For each case, the effect of filter coefficients' quantization in performance, in terms of Peak Signal to Noise Ratio (PSNR), is presented. Furthermore for both cases, the straightforward (multiplier-based) and a speed – area optimized implementation are compared.

Experimental results concerning the PSNR have been acquired by running a row – column implementation of the DWT in software, collaborating with a hardware implementation of the filter under test. The filter alternatives are mapped on an FPGA, using a prototyping platform hosting a Xilinx Virtex device. The results are general and independent from any normalization policy.

This work was supported by the project PENED '99 funded by G.S.R.T of Greek Ministry of Development.

II. THE LIFTING SCHEME APPROACH

The lifting scheme based DWT has been included in the upcoming JPEG2000 standard because it reduces the arithmetic complexity [4] of the conventional, convolution based DWT, up to a factor of two.

The lifting-based DWT implementation of filtering as described in [2] is given bellow. Applying the following steps to the entire input performs the transformation. The input is extended before and after the first and last coefficient, i_0 is the index of the first coefficient of the input and i_1 is the index of the coefficient immediately following the last coefficient.

Forward transformation

$Y_{2n+1} = X_{2n+1} + \alpha \times (X_{2n} + X_{2n+2})$ $\quad i_0-3 \le 2n+1 < i_1+3$ (S1)
$Y_{2n} = X_{2n} + \beta \times (Y_{2n-1} + Y_{2n+1})$ $\quad i_0-2 \le 2n < i_1+2$ (S2)
$Y_{2n+1} = Y_{2n+1} + \gamma \times (Y_{2n} + Y_{2n+2})$ $\quad i_0-1 \le 2n+1 < i_1+1$ (S3)
$Y_{2n} = Y_{2n} + \delta \times (Y_{2n-1} + Y_{2n+1})$ $\quad i_0 \le 2n < i_1$ (S4)
$Y_{2n+1} = -K \times Y_{2n+1}$ $\quad i_0 \le 2n+1 < i_1$ (S5)
$Y_{2n} = Y_{2n} / K$ $\quad i_0 \le 2n < i_1$ (S6)

Inverse transformation

$X_{2n} = K \times Y_{2n}$ $\quad i_0-3 \le 2n < i_1+3$ (S1)
$X_{2n+1} = - Y_{2n+1} / K$ $\quad i_0-2 \le 2n+1 < i_1+2$ (S2)
$X_{2n} = X_{2n} - \delta \times (X_{2n-1} + X_{2n+1})$ $\quad i_0-3 \le 2n < i_1+3$ (S3)
$X_{2n+1} = X_{2n+1} - \gamma \times (X_{2n} + X_{2n+2})$ $\quad i_0-2 \le 2n+1 < i_1+2$ (S4)
$X_{2n} = X_{2n} - \beta \times (X_{2n-1} + X_{2n+1})$ $\quad i_0-1 \le 2n < i_1+1$ (S5)
$X_{2n+1} = X_{2n+1} - \alpha \times (X_{2n} + X_{2n+2})$ $\quad i_0-1 \le 2n+1 < i_1$ (S6)

Where the values of the constants are:

$\alpha = -1.586134342$ $\quad \gamma = 0.882911075$
$\beta = -0.052980118$ $\quad \delta = 0.443506852$
$K = 1.230174105$

Each step represents a basic processing element. The steps S1 $\rightarrow$ S4 of the forward transformation and the last four steps, S3 $\rightarrow$ S6 of the inverse transformation, have the same structure and can be implemented by a processing block like the one displayed in Fig. 1 The rest steps are multiplications with constants (scaling steps).

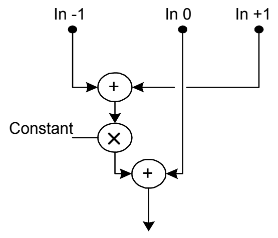


Fig.1 *Basic processing block of the lifting scheme.*

1. QUANTIZATION OF CONSTANTS.

The 9/7 filter is originally based on a floating point data representation. Since the scope here is hardware implementation of the 9/7 filter, we focus on the equivalent filter that uses fixed point (FXP) data representation. This is due to the fact that floating-point data-path operators are more complex, occupy more area and are slower than their FXP counterparts. The aim of this subsection is to define the effect of quantization of the FXP representation of samples and filters constants on image quality, in terms of PSNR, and filter implementation, in terms of speed and area.

Coding images using the DWT, requires a relatively small number of decomposition levels, e.g. for a 512×512 picture, 5 or 6 levels are enough. Thus the accuracy of the calculations may be decreased up to the point that the PSNR of the transformed and recovered image, ranges above a certain limit. This way, the bit-width of the filter data-path can be reduced, resulting to a decrease in memory requirements and smaller multipliers. The accuracy of the constants used in the calculations, can also be decreased, allowing also for the use of smaller multipliers.

After running the DWT using several gray-scale images with a color depth of 8 bits/pixel (the same color depth as that of the luminance component), several sizes and up to nine layers of transformation, it has been observed that a dynamic range of –2048 to 2047 is safely adequate. For a 2's complement FXP representation, this translates to 12 bits for the integer part. For the fractional part, 12 bits provide the accuracy to obtain a PSNR over 50dB for six levels of decomposition. So, a total of 24 bits is used for the data path.

The constants α, β, γ, δ, K, $^1/_K$ are quantized taking in account the number of bits with value '1', in their positive representation. That's because each '1' yields a term to be summed. The sets of constants, used to take the results shown in Table 1, are given bellow.

Set (A)

$\lvert\alpha\rvert$	= 01.1001011000001	$\lvert\gamma\rvert$	= 00.111000100000011
$\lvert\beta\rvert$	= 00.00001101100100000001	$\lvert\delta\rvert$	= 00.0111000110001
$\lvert K\rvert$	= 01.0011101011101	$\lvert ^1/_K\rvert$	= 00.1101000000011001

Set (B)

$\lvert\alpha\rvert$	= 01.1001011	$\lvert\gamma\rvert$	= 00.11100010000001
$\lvert\beta\rvert$	= 00.000011011001	$\lvert\delta\rvert$	= 00.011100011
$\lvert K\rvert$	= 01.0011101011	$\lvert ^1/_K\rvert$	= 00.1101000000011

The following results for the PSNR, achieved by different quantizations of the constants, are obtained using two test images: lena (512x512x8) and bridge (512x512x8). The forward filter is a hardware implementation using the constants above. For the inverse filtering, two type of filters are used. A software, double precision, filter with no quantization error and a hardware implementation, using the quantized constants.

		Levels					
Inverse Filter type	Image	1	2	3	4	5	6
software (double prec.)	lena	∞	∞	∞	∞	∞	81 dB
	bridge	∞	∞	∞	65 dB	58 dB	50 dB
Hardware impl.	lena	∞	∞	∞	∞	73 dB	66 dB
	bridge	∞	∞	∞	∞	71 dB	53 dB

Table 1a. *PSNR measurements for set of constants (A).*

		Levels					
Inverse Filter type	Image	1	2	3	4	5	6
software (double prec.)	lena	∞	∞	∞	∞	∞	82 dB
	bridge	∞	∞	∞	65 dB	58 dB	50 dB
Hardware impl.	lena	∞	∞	∞	53 dB	49 dB	48 dB
	bridge	∞	∞	∞	50 dB	49 dB	47 dB

Table 1b. *PSNR measurements for set of constants (B).*

2. ARCHITECTURES FOR THE LIFTING-SCHEME PROCESSING UNITS.

In the following, the basic processing elements of the lifting scheme will be examined.

(i) Multiplier – based.

A straightforward implementation for the lifting scheme uses the processing unit shown in Fig. 1 with a multiplier capable to handle signed numbers and two adders for each processing unit. Providing the appropriate constant to the multiplier, implements the desired lifting step. The width of the multipliers is determined by the accuracy of the constants and the data path bit-width.

The drawback of the above implementation is that the multipliers occupy a great amount of area and restrict the throughput of the processing unit.

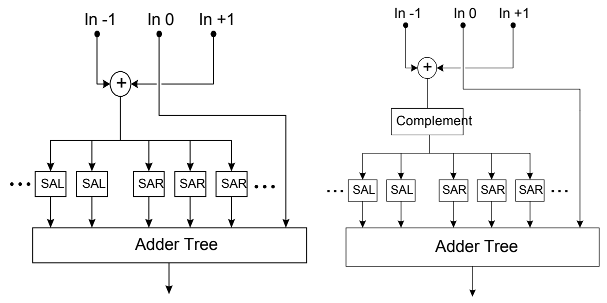
(ii) Optimized, using shift-add operations.

Using shift-add operations to replace the multiplications with constants optimizes the above implementation. An improved processing block can be obtained that way, but a separate block is needed to perform the multiplication with each constant. The architecture of the optimized processing unit is shown in Fig. 2. A comparison can be made by examining the results given at the end of this section, in Table 2.

Two different architectures are given, depending on the constant's sign. The multiplication with a positive constant is translated in summing shifted versions of the input. When the positive constant is in FXP format, a term corresponds to each bit with value '1'.

For example, multiplication with the constant 2.25, which is represented in FXP format as 0010.0100 equivalents in adding two terms. The first term is the input shifted arithmetically left for one position and the second term is the input shifted arithmetically right two positions.

When the constant is negative, it is represented as positive and the sign of the input is complemented. When the input is in 2's complement FXP format, this is done by inverting the input and adding one least significant bit.



(a) **(b)**

Fig. 2 *Basic processing block for:*
(a) *positive constants.*
(b) *negative constants.*

The following experimental results, show the area – delay characteristics for each processing unit, for the forward filtering. The results for the inverse filter processing units do not differ significantly and aren't given.

	Processing Unit											
Forw.	a		b		c		d		k		1/k	
Set (A)	100 CLBs	17 ns	90 CLBs	18 ns	72 CLBs	13 ns	73 CLBs	13 ns	106 CLBs	15 ns	49 CLBs	9 ns
Set (B)	94 CLBs	17 ns	86 CLBs	17 ns	69 CLBs	13 ns	67 CLBs	13 ns	87 CLBs	13 ns	44 CLBs	9 ns

Table 2 *Area – delay results for the shift - add architecture.*

III. CONVOLUTIONAL FILTERS

The convolutional-based DWT uses two FIR filters, one for calculating the low-pass coefficients and one for the high-pass coefficients. The following sets of coefficients implement a filtering unit equivalent – interchangeable with the lifting scheme filtering units described above.

<u>Forward filters</u>

$$\begin{aligned} Y_{2n} = {} & cl0 \cdot X_{2n} + \\ & cl1 \cdot (X_{2n-1}+X_{2n+1}) + \\ & cl2 \cdot (X_{2n-2}+X_{2n+2}) + \\ & cl3 \cdot (X_{2n-3}+X_{2n+3}) + \\ & cl4 \cdot (X_{2n-4}+X_{2n+4}) \end{aligned}$$

$$\begin{aligned} Y_{2n+1} = {} & ch0 \cdot X_{2n+1} + \\ & ch1 \cdot (X_{2n}+X_{2n+2}) + \\ & ch2 \cdot (X_{2n-1}+X_{2n+3}) + \\ & ch3 \cdot (X_{2n-2}+X_{2n+4}) \end{aligned}$$

$cl0 = 0.602\,949\,018\,236$
$cl1 = 0.266\,864\,118\,442$
$cl2 = -0.078\,223\,266\,528$
$cl3 = -0.016\,864\,118\,442$
$cl4 = 0.026\,748\,757\,410$

$ch0 = -1.115\,087\,052\,456$
$ch1 = 0.591\,271\,763\,114$
$ch2 = 0.057\,543\,526\,228$
$ch3 = -0.091\,271\,763\,114$

<u>Inverse filters</u>

$$\begin{aligned} X_{2n} = {} & cl0 \cdot Y_{2n} + \\ & cl1 \cdot (Y_{2n-1}+Y_{2n+1}) + \\ & cl2 \cdot (Y_{2n-2}+Y_{2n+2}) + \\ & cl3 \cdot (Y_{2n-3}+Y_{2n+3}) + \end{aligned}$$

$$\begin{aligned} X_{2n+1} = {} & ch0 \cdot Y_{2n+1} + \\ & ch1 \cdot (Y_{2n}+Y_{2n+2}) + \\ & ch2 \cdot (Y_{2n-1}+Y_{2n+3}) + \\ & ch3 \cdot (Y_{2n-2}+Y_{2n+4}) + \\ & ch4 \cdot (Y_{2n-3}+Y_{2n+5}) \end{aligned}$$

$cl0 = 1.115\,087\,052\,456$
$cl1 = 0.266\,864\,118\,442$
$cl2 = -0.057\,543\,526\,228$
$cl3 = -0.016\,864\,118\,442$

$ch0 = -0.602\,949\,018\,236$
$ch1 = 0.591\,271\,763\,114$
$ch2 = 0.078\,223\,266\,528$
$ch3 = -0.091\,271\,763\,114$
$ch4 = -0.026\,748\,757\,410$

1. QUANTIZATION OF COEFFICIENTS.

As stated before, the data path width is 24 bits. Each coefficient of the filter is quantized with accuracy proportional to its value, because larger coefficients affect the filter output more than smaller ones. Two coefficient sets with different accuracies are given next.

<u>Forward Set (A)</u>

cl0 = 00.1001101001011
cl1 = 00.010001000101
cl2 = 11.111011
cl3 = 11.111110111011
cl4 = 00.0000011011011

ch0 = 10.111000101001
ch1 = 00.100101110101
ch2 = 00.000011101011
ch3 = 11.111010001011

<u>Inverse Set (A)</u>

cl0 = 01.00011101011
cl1 = 00.010001000101
cl2 = 11.111100010101
cl3 = 11.111110111011

ch0 = 11.011001011011
ch1 = 00.1001011101011
ch2 = 00.000101
ch3 = 11.1110100010101
ch4 = 11.111110010011

<u>Forward Set (B)</u>

cl0 = 00.100110100101
cl1 = 00.010001000101
cl2 = 11.111011
cl3 = 11.1111101111
cl4 = 00.000001101101

ch0 = 10.111000101001
ch1 = 00.100101110101
ch2 = 00.000011101011
ch3 = 11.111010001011

<u>Inverse Set (B)</u>

cl0 = 01.000111010111
cl1 = 00.010001000101
cl2 = 11.111100010101

ch0 = 11.011001011011
ch1 = 00.100101110101
ch2 = 00.000101

cl3 = 11.1111101111   ch3 = 11.111010001011
ch4 = 11.111110010011

In the following tables, the PSNR values achieved for the two different coefficient sets are given. The test image bridge (512x512x8) was used.

	Levels					
Inverse Filter type	1	2	3	4	5	6
software (double precision)	∞	∞	∞	∞	∞	73 dB
Hardware implementation	∞	∞	∞	93 dB	79 dB	68 dB

Table 3a. *PSNR for quantized set of coefficients (A).*

	Levels					
Inverse Filter type	1	2	3	4	5	6
software (double precision)	∞	∞	99 dB	69 dB	62 dB	58 dB
Hardware implementation	∞	102dB	67 dB	57 dB	54 dB	52 dB

Table 3b. *PSNR for quantized set of coefficients (B).*

2. THE FIR FILTER IMPLEMENTATION.

(i) Multiplier – based.

A straightforward architecture is the one shown in Fig. 3 bellow. For this implementation, nine multipliers and fourteen adders are needed.

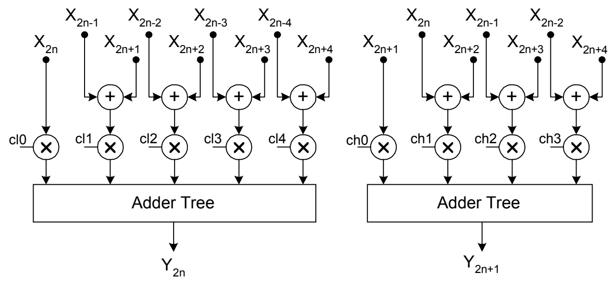


Fig. 3 *Architecture of the forward convolutional filters with multipliers.*

(ii) Optimized, adder – based.

Using the technique described before in the lifting scheme approach, the multipliers are reduced to shift – add operations. This leads to a more compact and slightly faster implementation than the previous one, which used multipliers. The block diagram for the low pass filter is shown in Fig. 4. The high pass filter is similar.

The area and delay results for the two architectures and for the coefficient sets given above are presented in Table 4. The results were taken by implementing on a Xilinx XCV300-5 FPGA.

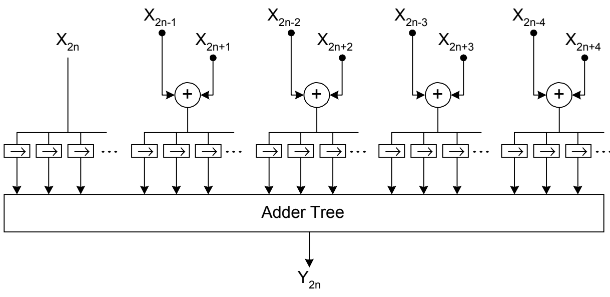


Fig. 4 *Architecture of the forward L.P. convolutional filter.*

Architecture	Coef. Set	Filter type	Area (CLB Slices)	Delay (ns)
Multipliers	Set (A)	Forw	1801	24.4
		Inv	1787	24.4
	Set (B)	Forw	1745	24.2
		Inv	1745	24.3
Shift -Add operations	Set (A)	Forw	634	19.3
		Inv	657	19.7
	Set (B)	Forw	604	18.7
		Inv	620	20.6

Table 4 *Comparative results between the two architectures.*

IV. CONCLUSIONS

By comparing the results for the optimized architectures of set (B) from Table 4 with those of set (A) from Table 2, which achieve similar PSNR results, we observe that the six processing units for the lifting scheme occupy about 20% less area than the convolutional filters. In lifting scheme, each processing unit depends on results from another processing unit, so they cannot operate in parallel, unless a form of pipelining is used. The total delay for one filtering operation with lifting scheme is larger than the convolutional filter's delay. It seems that for hardware implementation the optimized convolutional approach is faster, with about the same area than the lifting scheme approach, and requiring much simpler control logic.

REFERENCES

[1] M. Antonini, et al. , "Image Coding Using Wavelet Transform" IEEE Transactions on Image Processing, Vol. 1, No. 2, pp. 205-220, April 1992.

[2] ISO/IEC FCD15444-1: 2000 V1.0, "*JPEG 2000 Image Coding System* ", official release expected at Mar. 2001

[3] ISO/IEC JTC1/SC29/WG11, FCD 14496-1, "*Coding of Moving Pictures and Audio*", May 1998.

[4] W. Sweldens, "*The lifting scheme: A construction of second generation wavelets*", in SIAM J. Math. Anal., no 2, Vol. 29, pp. 511-546, 1997.