VLSI Implementation of 2-D DWT/IDWT Cores using 9/7-tap filter banks
based on the Non-expansive Symmetric Extension Scheme

Kavish Seth and S. Srinivasan
Department of Electrical Engineering
Indian Institute Technology, Madras, Chennai-600036, India
{ee99m06, srini}@ee.iitm.ernet.in

Abstract

This paper presents architectures and scheduling algorithms for the 2-D Discrete Wavelet Transform (DWT) and the Inverse Discrete Wavelet Transform (IDWT) using 9/7-tap filter banks based on the Non-expansive Symmetric Extension (NSE) scheme that reduces distortion at boundaries of reconstructed image. The hardware has been implemented for the image blocks of size 32 x 32 pixels, up to third level of transform and cuts down the power consumption at architecture level by incorporating three techniques, viz., Canonic Sign Digit (CSD) and common subexpression sharing technique, Gray code addressing mode and resource sharing. The implementation has been tested using 0.35 μm (three metal) technology by simulation at functional, circuit and physical levels. The performance measures of implementation, viz., area, memory requirement, speed and power have been evaluated.

I. INTRODUCTION

The Discrete Wavelet Transform (DWT) of a signal [1] is an alternative to the existing time-frequency representations such as DFT and DCT, and is becoming popular in many signal processing applications. It can be used as a viable tool especially in image compression because of its capability for multi-resolution representation. For example, a new entropy coding technique has been proposed in [2] and its performance has been shown to be competitive with all known techniques with respect to compression ratio and image quality.

There exists a large number of architectures for computing 1-D and 2-D DWT [3]-[7]. All these architectures assume zero padding so that the first coefficient of level 1 can be computed using only the first coefficient of level 1-1. While this results in smaller latency, the resulting coefficients can not be used to obtain perfect reconstruction. In order to obtain perfect reconstruction, a periodic extension of the signal or a symmetric extension of the signal has to be used. The method based on symmetric extension is chosen for low bit rate coding as it gives the improved performance.

This is because the pixel intensities at opposite boundaries are vastly different and a periodic extension of the signal may result in distortion at the boundaries, which can not be coded well by low bit rate coders.

In this paper, we propose architectures and scheduling algorithms for 2-D-DWT/IDWT modules based on the Non-expansive Symmetric Extension (NSE) scheme [8]. First, the architectures are folded as per [7] and then modified for the NSE scheme. The motivation for applying the NSE scheme is its two main advantages, viz., it reduces the distortion at boundaries of a reconstructed image and is suitable for low bit rate coders [13]. Power consumption is reduced at architecture level by incorporating three techniques, viz., Canonic Sign Digit (CSD) and common subexpression sharing technique [9], Gray code addressing mode in all memory decoders and resource sharing [11].

In this paper, our main focus is on single chip implementation of 2-D DWT/IDWT modules. Though this implementation is dedicated to 9/7-tap filter banks proposed in [14] and the image block of size 32x32 pixels, the proposed architectures and algorithms are generalized and can be used for any length of filter banks, any size of images and any level of transform.

The rest of paper is organized as follows. Subband structure of the encoder and decoder, the basics of the NSE methods and a brief introduction of CSD arithmetic and common sub-expression technique are discussed in Section II. Architectures and scheduling algorithms for 2-D DWT and 2-D IDWT modules are proposed in section III. Section IV describes the implementation and simulation results and conclusions are summarized in Section V.
II. PRELIMINARIES

The Wavelet Transform

Two dimensional DWT and IDWT modules can be implemented by the filter bank structures shown in Figures 1 (a) and 1 (b) respectively. In Figure 1 (a), the four channel subband decomposition of 2-D DWT is obtained by separable applications of the two channel decomposition of 1-D DWT in the horizontal and vertical dimensions. Similarly, 2-D IDWT is obtained by two channel reconstruction of 1-D IDWT along the horizontal and vertical dimensions. For these implementations, it is assumed that (i) the image size is $N \times N$, (ii) the number of levels is $J$ and (iii) the low pass and the high pass filters are symmetric filters of sizes $K_1$ and $K_2$ respectively.

In the 2-D DWT filter bank structure, the low pass and the high pass outputs after $(a+1)$th stage computations along row are denoted by $(LL)^a \ L$ and $(LL)^a \ H$. The size of the $(LL)^a \ L$ and $(LL)^a \ H$ output arrays is $N/2^a \times N/2^a$, $0 \leq a < J$, $(LL)^a \ LH$ and $(LL)^a \ LL$ are the high pass and the low pass outputs after computing along the columns of $(LL)^a \ L$, while $(LL)^a \ HH$ and $(LL)^a \ HL$ are the high pass and low pass outputs after computing along the columns of $(LL)^a \ H$. The size of the $(LL)^a \ HH$, $(LL)^a \ HL$, $(LL)^a \ LH$ and $(LL)^a \ LL$ output arrays is $N/2^{a+1} \times N/2^{a+1}$, $0 \leq a < J-1$.

In the 2-D IDWT filter bank structure shown in Figure 1 (b), column computations are used to reconstruct $(L'L')^a \ L'$ from $(L'L')^a \ H'$ and $(L'L')^a \ L'$. Similarly, column computations are used to reconstruct $(L'L')^a \ H'$ from $(L'L')^a \ H'$ and $(L'L')^a \ H'$. The size of the $(L'L')^a \ L'$ and $(L'L')^a \ H'$ arrays is $N/2^a \times N/2^a$, $0 \leq a < J$. Row computations are used to reconstruct $(L'L')^a$ outputs from $(L'L')^a \ H'$ and $(L'L')^a \ L'$. The size of the $(L'L')^a$ outputs is $N/2^a \times N/2^a$, $0 \leq a < J$.

The Non-expansive Symmetric Extension Scheme

There are two distinct classes of linear phase finite impulse response (FIR) filters. The first class will contain pairs of odd-length, symmetric filters (i.e., filters whose impulse responses are symmetric about their middle sample). These are called “Type I” linear phase FIR filters [12], or Whole-Sample Symmetric (WSS) filters. The second class will contain pairs of even-length filters, one symmetric (the low pass filter) and one anti-symmetric (the high pass filter). These are called “Type II” and “Type IV” filters respectively [12], since such filters are symmetric about the point halfway between the middle samples.
between their middle two samples. They are also referred as Half-Sample Symmetric (HSS/HAS or HS-type) filters.

Consider a 1-D input signal \( x(n) \) of length \( N_0 \). The leading edge of the signal is symmetrically extended by reflecting the first few samples and trailing edge is similarly extended by the last few samples. The type of extension depends upon the type of filter and the length of signal. For example, in analysis bank, (1,1)-symmetric extension is used for WSS filters and (2,2)-symmetric extension is used for HS filters. The synthesis filters have to operate on symmetric extended signals as well to get the perfect reconstruction [8].

In this paper, we present the implementation for 9/7-tap filters which are of WSS type and also assume that \( N_0 = 32 \). For such type of configuration, we use (1,1)-symmetric extension in analysis bank and (1,2)-symmetric extension for low pass filters and (2,1)-symmetric extension for high pass filters in synthesis bank. However, architectures and scheduling algorithms have been generalized for all types of symmetric extension schemes.

\section*{CSD Arithmetic and Sub-expression Sharing}

The number of add operations required in a constant coefficient multiplication equals one less than the number of nonzero bits in the constant coefficient. In order to further reduce the area and power consumption, the constant coefficient can be encoded such that it contains the fewest number of nonzero bits, which can be accomplished using Canonic Signed Digit (CSD) representation. The properties of CSD number representations and algorithm for computing CSD format have been described in [9]. Constant multiplication is carried out by adding or subtracting the number of partial product terms corresponding to the nonzero bit positions in the constant multiplier. A CSD-encoded multiplier contains the least number of nonzero bits and, hence, requires the minimum number of addition/subtraction operations [9].

If the multiplier is represented in Canonic Signed Digit (CSD) form, then the number of addition (or subtraction) operations will be minimum. In the case where several multipliers are present in a network of operators, for instance an FIR filter, the savings achieved by identifying common sub-expressions can be as much as 50% of the total number of operators [10]. The common sub-expression sharing technique is best suited within individual filter coefficient as well as within two filter coefficients. However, in the present implementation, the sub-expressions can be shared only within the individual filter coefficient because none of the filters used for this implementation has transpose for transposed structure of FIR filters. In transposed structure of FIR filters, the sub-expressions can be shared structure. This scheme is not as effective as [10] but it is still able to reduce the number arithmetic operations to some extent.

\section*{III. ARCHITECTURE AND SCHEDULING ALGORITHM FOR 2-D DWT AND 2-D IDWT}

\subsection*{2D-DWT}

The proposed folded architecture consists of two parallel computational units which compute along the rows, two parallel computational units which compute along the columns, and two storage units. Figure 2 gives the block diagram of this architecture.

Here, Filters 1 and 2 are two parallel computational units based on NSE scheme, which compute along the rows. Filters 3 and 4 are parallel computational units, which compute along the columns. The outputs of Filters 1 and 2 are stored in symmetric extension register bank -1 (SERB-1) in row major order and are read out along the columns by Filters 3 and 4.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{architecture.png}
\caption{The Block Diagram of the Architecture for 2-D DWT}
\end{figure}

NSE is performed in a special type of shift register, called NSE register. The length of NSE register is equal to the length of the filter with larger length. The architecture of the NSE register is shown in Figure 3. The NSE register works in the following three modes, viz., initial extension, normal and final extension. In the initial extension mode, data is routed with the help of multiplexers. In the normal mode, it works as a normal shift register. In the final extension mode, some of the outputs of the NSE register are multiplexed with other outputs in the Final Extension Block (FEB) to get the required extension. The FEB is transparent for the NSE register outputs of initial extension and normal modes. The outputs of the NSE register in different modes are shown in Figure 4. The data routing in the NSE register is controlled by the Router Control Logic Unit (RCLU). We next describe the operations in each filter and each storage unit.
**Filter 1** is an NSE scheme based filter and computes the low pass and high pass outputs (L and H) along each row. L are computed every other cycle. L and H are computed in an interleaved fashion.

**Filter 2** computes the outputs (LL)^a L and (LL)^a H along the rows, 1 ≤ a < J. It first computes the outputs (LL)^a L and N cycle later, computes the output (LL)^a H. This filter reads the same data from SERB-II twice, since the inputs for the computation of (LL)^a L and (LL)^a H are the same.

**Filter 3** computes the outputs (LL)^a LL and (LL)^a HL along the columns, 0 ≤ a < J. It first computes the outputs (LL)^a LL N cycles later, computes the outputs of (LL)^a HL.

**Filter 4** computes the outputs (LL)^a LH and (LL)^a HH along the columns, 0 ≤ a < J. It first computes the outputs of (LL)^a LH and N cycle later, computes the outputs of (LL)^a HH.

**SERB-I** has different types of the NSE registers. The type depends upon the value of a, 0 ≤ a < J. (LL)^a L_d is stored in the L^a_d register and (LL)^a H_d is stored in the H^a_d register, 0 ≤ i ≤ N/2, 0 ≤ j ≤ N/2^a+1. The length of all the NSE registers in SERB-I are same and equal to max(K1, K2).

**SERB-II** has total J of different types of the NSE registers. The type depends upon the value of a, 1 ≤ a < J. The outputs of Filter 3 (LL)^a LL are stored in the NSE register LL^a. Filter 2 reads the same data from SERB-II twice, since the inputs for the computation of (LL)^a L and (LL)^a H are the same. This means, in order to compute (LL)^a H, we have to push the same data in LL^a register once again. To maintain continuity of the same data, we use the shift register bank (SRB). It has total J-1 different length of the shift registers. A register in SRB represented as SR-a has length of N/2^a. 1 ≤ a < J. While computing (LL)^a L, (LL)^a LL are pushed simultaneously in the NSE register as well as the shift register (SR-a) and to compute (LL)^a H, data is pushed in the NSE register from SR-a.

The scheduling algorithm for this architecture is described below.

1. Schedule L_d and H_d in interleaved fashion with Filter 1 and store in L^a_d and H^a_d registers respectively, 0 ≤ i ≤ N and 0 ≤ j ≤ N/2.
2. For 1 ≤ a < J, if initial extension is over in the register L^a_d, Filter 3 and Filter 4 start computing (LL)^a LL and (LL)^a LH simultaneously, 0 ≤ i ≤ N/2^a. (LL)^a LL_d is stored in the LL^a_d register as well as SR-a. (LL)^a HL_d is scheduled 2^a cycles after (LL)^a LL_d. In the same way (LL)^a LH_d is computed.

3. For 2 ≤ a < J schedule (LL)^a LL_d such that there should not be any conflict with (LL)^a LL_d. 1 ≤ b < a. If there is any conflict, shift the scheduling of (LL)^a LL_d for minimum number of clock cycles so that the conflict can be avoided.

4. For 1 ≤ a < J, Filter 3 and Filter 4 start computing (LL)^a HL_d and (LL)^a HH_d, N cycles after (LL)^a LL_d, 0 ≤ i ≤ N/2^a. (LL)^a HL_d is scheduled 2^a cycles after (LL)^a HH_d. In the same way (LL)^a HH_d is computed.

5. For 1 ≤ a < J, if initial extension is over in the LL^a register, Filter 2 starts computing (LL)^a L_d, 0 ≤ i ≤ N/2^a, 0 ≤ j ≤ N/2^a. (LL)^a H_d is scheduled N cycles after (LL)^a L_d. (LL)^a L_d and (LL)^a H_d are stored in L^a_d and H^a_d registers respectively.

**2-D IDWT**

The architecture of 2D-IDWT consists of two parallel computational units, which compute along the rows (Filters 1 and 2), two parallel computational units, which compute along the columns (Filters 3 and 4), and two storage units as shown in Figure 5. The input is read from
Figure 4. The NSE Register Outputs in different modes for 9/7-tap Analysis Filters

Filter 3 and Filter 4. Filters 1 and 2 compute along the columns and store the outputs in Symmetric Extension Register Bank (SERB) unit in the row major order. Filter 3 and Filter 4 read the data out of SERB along the rows and write the outputs in Storage Bank (SB) in row major order.

A scheduling algorithm, similar to the one proposed for DWT, has also been proposed for this architecture. The outputs of the NSE registers for low pass and high pass filters in different modes for this architecture are shown in Figures 6 (a) and 6 (b) respectively.

IV. RESULTS AND DISCUSSION

The DWT/IDWT cores have been implemented using 0.35 μm, 3.3 volt CMOS technology with three layers of metal. Synopsys tools have been used for the front-end steps, which capture, simulate and synthesize the various schemes given in Table 1. Cadence tools have been used to generate final layout of NSE based 2D-DWT module with CSD multipliers by using gate-level Verilog netlist.

Table 1 compares the performances of various DWT/IDWT implementations. It is clear from the this table that in spite of having small area and low memory requirements with low power consumption, the zero-packing scheme for image blocks of 32x32 pixels gives very low PSNR value for 256x256 pixels Lena image and hence is not useful for practical applications. However, the zero-packing scheme for image blocks of 256x256 pixels gives a good PSNR value for the same image at the expense of large memory requirement with very high power consumption and hence is also not suitable for VLSI implementation. The NSE scheme with normal multipliers for image blocks of 32x32 pixels gives a better PSNR value than the above mentioned schemes. However, compared to the Zero-packing scheme with normal multipliers for 32x32 block size, it requires the same amount of memory, but more area because of extra NSE logic and its power consumption is also almost double. The power consumption and area are reduced in the NSE scheme with CSD multipliers at the expense of slight reduction in quality. This scheme gives

45.12 dB PSNR value for 256x256 pixels Lena image and this value is quite reasonable for good quality of reconstructed images. The maximum frequency of operation is also higher than the other implementations.

Figure 5. The Block Diagram of the Architecture for 2-D IDWT

![Diagram](image)

Figure 6. The NSE Register Outputs for (a) 9-tap High Pass (b) 7-tap Low Pass Synthesis Filters

![Diagram](image)
<table>
<thead>
<tr>
<th>Type of scheme</th>
<th>Area (mm²)</th>
<th>Power at 20 MHz (mW)</th>
<th>Maximum Frequency of Operation (MHz)</th>
<th>Memory Requirement (KB)</th>
<th>PSNR with 256x256 pixels Lena image (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero-pulsing Scheme with Normal Multipliers for 32x32 pixels block size</td>
<td>5.05</td>
<td>4.26</td>
<td>19.85</td>
<td>20.69</td>
<td>50</td>
</tr>
<tr>
<td>Zero-pulsing Scheme with Normal Multipliers for 250x256 pixels block size</td>
<td>23.44</td>
<td>19.78</td>
<td>153.04</td>
<td>160.51</td>
<td>50</td>
</tr>
<tr>
<td>Non-expansive Symmetric Extension Scheme with Normal Multipliers for 32x32 pixels block size</td>
<td>6.63</td>
<td>6.92</td>
<td>38.59</td>
<td>49.03</td>
<td>50</td>
</tr>
<tr>
<td>Non-expansive Symmetric Extension Scheme with CSD Multipliers for 32x32 pixels block size</td>
<td>5.48</td>
<td>6.05</td>
<td>30.06</td>
<td>38.70</td>
<td>70</td>
</tr>
</tbody>
</table>

Table 1: The Comparison of various DWT/IDWT (3-Level) Implementations

V. CONCLUSIONS

The NSE based DWT/IDWT architectures reduce the distortion at boundaries of a reconstructed image and are also suitable for low bit rate coders. Though this implementation is dedicated to 9/7-tap filter banks and an image block of size 32x32 pixels, the proposed architectures and algorithms are generalized and can be used for any length of filter banks, any size of images and any level of transform. The implementation has been tested for image block of size 32 x 32 pixels up to third level of transform and found to work satisfactorily. Any image whose dimensions are multiples of 32, can be fed as input to DWT/IDWT modules. Both the cores operate in burst mode.

REFERENCES