A DWT-BASED ENCODER ARCHITECTURE FOR SYMMETRICALLY EXTENDED IMAGES

Chaitali Chakrabarti

Department of Electrical Engineering
Telecommunications Research Center
Arizona State University
Tempe, AZ 85287-5706
chaitali@asu.edu

ABSTRACT

In this paper we present an architecture for a 2-D Discrete Wavelet Transform (DWT)-based encoder that handles computations along the border efficiently by using the method of symmetric extension. We choose symmetric extension (SA) as opposed to zero padding or periodic extension since (i) the coefficients generated by SA can be used to obtain perfect reconstruction and (ii) it is better suited for low bit rate coders. The proposed architecture is similar to the existing 2-D DWT architectures for zero padded images with the notable exception that additional router units are now required to reorder the data. Reordering is essential to sustain the computation of successive problem instances in symmetrically extended images. We show that the router can be implemented using simple combinational logic, resulting in minimal area overhead.

1. INTRODUCTION

The last few years have seen wavelet transforms being used for a wide range of applications, including, signal analysis, signal coding, image processing, data compression, numerical analysis, statistics and physics. The wavelet transform can be viewed as a decomposition of a signal in the time-scale plane. There are several types of wavelet transforms depending on the nature of the signal (continuous or discrete) and the nature of the time and scale parameters (continuous or discrete). In this paper we focus on the realization of the 2-dimensional (2-D) Discrete Wavelet Transform (DWT), where the signal as well as the time and scale parameters are discrete.

There now exists a large number of architectures for computing 1-D and 2-D DWT [1]-[5]. These architectures range from systolic array/parallel filter implementations to SIMD arrays. All these architectures assume zero padding so that the first coefficient of level i can be computed using only the first coefficient of level $i-1$. While this results in smaller latency, the resulting coefficients cannot be used to obtain a perfect reconstruction of the original signal. To obtain perfect reconstruction, a periodic extension of the signal or a symmetric extension of the signal has to be used. We choose the method based on symmetric extension since for low bit rate coding, it gives improved performance [6]. This is because in images where the pixel intensity at opposite boundaries is greatly different, a periodic extension of the signal may result in distortion at the boundaries which cannot be coded well by low bit rate coders.

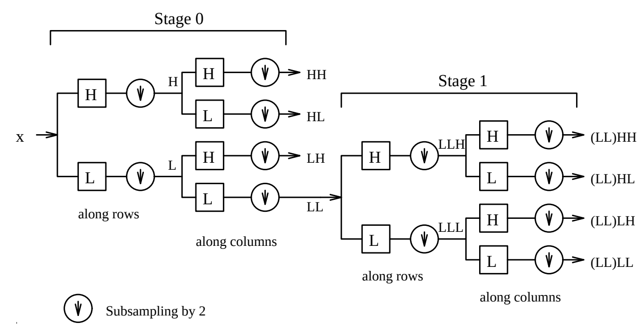

Figure 1. The 2-D DWT filter bank structure for $J = 2$ stages.

In this paper we present an architecture for 2-D DWT for the case when the signal is symmetrically extended. The proposed architecture is similar in esence to the existing 2-D DWT architectures for zero padded images [4]. The main difference is that additional router units are now needed at the inputs of the filters to reorder the data. Since successive problem instances of symmetrically extended images overlap in time, routing is essential to sustain the pipelined execution. The other difference is in the schedule (and consequently, the control logic) – the outputs can no longer be computed after only one input has been read in (as in the architecture for zero padded images).

The rest of the paper is organized as follows. In Section 2, we briefly describe the subband structure of the encoder and the basics of symmetric extension. We then describe the architectures that support symmetric extension – the 1-D DWT architecture in Section 3 and the 2-D DWT architecture in Section 4.

2. PRELIMINARIES

The encoder can be implemented by the filter bank structure shown in Figure 1. In this figure, 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. Let the image be of size $N \times N$ and the number of octaves be J. We denote the high pass and the low pass outputs after the $(a+1)$th stage computations along the row by $(LL)^a H$ and $(LL)^a L$. The size of the $(LL)^a H$ and $(LL)^a L$ output arrays is $\frac{N}{2^a} \times \frac{N}{2^{a+1}}$, $0 \le a < J$. $(LL)^a HH$ and $(LL)^a HL$ are the high pass and low pass outputs after computing along the columns of $(LL)^a H$, while $(LL)^a LH$ and $(LL)^a LL$ are the high pass and low pass outputs after computing along the columns of $(LL)^a L$. The size of the $(LL)^a HH$, $(LL)^a HL$, $(LL)^a LH$ and $(LL)^a LL$ output arrays is $\frac{N}{2^{a+1}} \times \frac{N}{2^{a+1}}$, $0 \le a < J-1$.

Symmetric extension: Consider a 1-D input signal $x(n)$ of length N. The leading edge of the signal is symmetrically extended by reflecting the first few samples; the trailing edge is similarly extended by reflecting the last few samples [7]. For even length filters, with L a symmetric filter and H an antisymmetric filter, the signal is symmetrically extended on the left by $K/2-1$ samples to create a sequence with half sample symmetry and on the right by $K/2-1$ samples to create a sequence with half sample symmetry. The resulting signal $\tilde{x}(n)$ is given by $x_{K/2-1}$, ..., x_0, x_0, x_1, ..., x_{N-2}, x_{N-1}, x_{N-1}, ..., $x_{N-K/2+1}$. For odd length filters, with L a symmetric filter of (say) length K_1, and H an antisymmetric filter of (say) length K_2, $K_1 > K_2$, the signal is symmetrically extended on the left by $\lfloor K_1/2 \rfloor$ samples to create a subsequence with whole sample symmetry and on the right by $\lfloor K_1/2 \rfloor$ samples to create a subsequence with whole sample symmetry. The resulting signal $\tilde{x}(n)$ is $x_{\lfloor K_1/2 \rfloor}$, ..., x_1, x_0, x_1, ..., x_{N-2}, x_{N-1}, x_{N-2}, ..., $x_{N-\lfloor K_1/2 \rfloor}$. Note that for perfect reconstruction, the synthesis filters have to operate on symmetrically extended signals as well. For more details, see [7].

3. 1-D DWT ARCHITECTURE

The proposed 1-D DWT architecture is an extension of the parallel filter architecture in [3]. It consists of two parallel filters to compute the low pass and the high pass outputs, a storage unit of size KJ to store the inputs that are required for the computation of the J octave outputs, and a router to reorder the data for symmetric extension. Note that the main architectural components are the same as in [3]. For even-length filters, only a single router is required, whereas for odd-length filters since the lengths of the high pass and the low pass filters are different, two routers are required. Figure 2 gives the block diagram of this architecture for even and odd-length filters. In the rest of this section, we focus primarily on even-length filters.

3.1. Schedule

The scheduling algorithm for symmetrically extended signals can be derived in a manner very similar to that of zero padded signals in [3]. For even-length filters, the main difference between the schedule used here and that used in [3] is that here the computations of the first output can start only after $K/2+1$ inputs have been read in (compared to only after one input has been read in [3]). If T is the latency of the parallel filters, and if the first input x_0 is input at time 0, then for even K, the various output octaves are computed at the following times.

First octave outputs. $y_1(i)$ at $2i + T + K/2$, $0 \le i < N/2$.

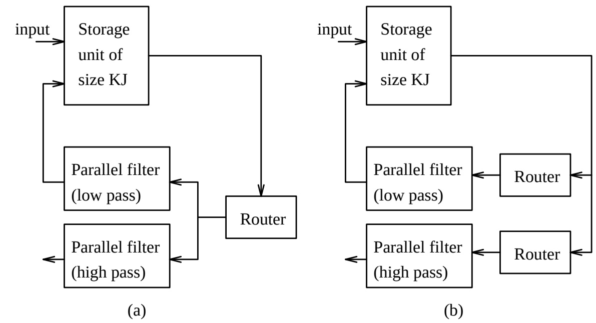

Figure 2. Block diagram of the parallel filter architecture for 1-D DWT for (a) even and (b) odd-length filters.

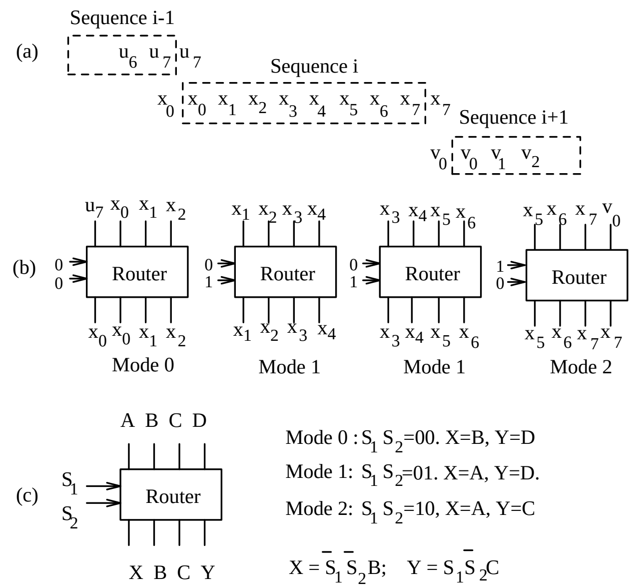

Figure 3. Example $N=8$, and $K=4$. (a) Symmetric extension (b) Router configurations and (c) Design of the router

Second octave outputs. $y_2(i)$ at $4i + 2T + 3K/2 + \beta_2$, $0 \le i < N/4$,

Third octave outputs. $y_3(i)$ at $8i + 3T + 7K/2 + \beta_2 + \beta_3$, $0 \le i < N/8$.

In general. $y_a(i) = 2^a i + aT + K/2 \sum_{i=0}^{a-1} 2^i + \sum_{i=2}^{a} \beta_i$, $1 \le a \le J$.

The β values are offsets that ensure that there is no conflict with the lower octave outputs.

3.2. Router design

The router reorders the data that is read from the storage unit and feeds it to the two parallel filters. Since the reordering is a function of time, a master controller is required to set the values of the control signals in each time instant. We explain the operation of the router with an example.

Example: For $N=8$ and $K=4$, the symmetrically extended data is $\tilde{x}(n) = \{x_0, x_0, x_1, \ldots, x_6, x_7, x_7\}$. In Figure 3(a), we show three symmetrically extended signals, $\tilde{u}$,

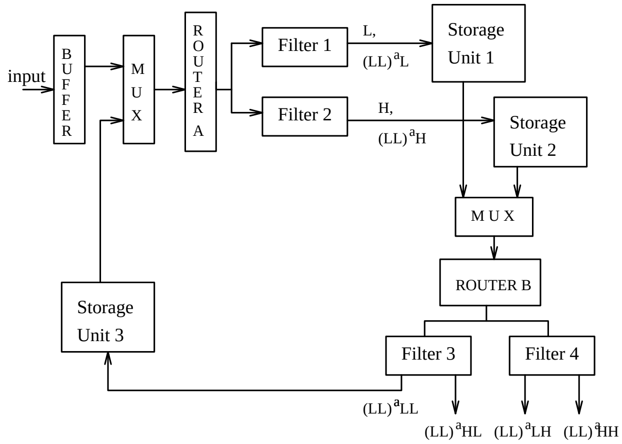

Figure 4. Block diagram of the architecture for 2-D DWT for even-length filters

$\tilde{x}$, and $\tilde{v}$. The actual signals, u, x and v are sent one after the other (for eg., x_0 is sent 1 cycle after u_7). While the actual signals do not overlap in time, their symmetric extensions do. Thus a router is essential for sustaining the pipelined execution.

The router operates in 3 modes: Mode 0 corresponds to symmetric extension on the left and is used for computing y_0, Mode 1 corresponds to normal operation and is used for computing y_1 and y_2, Mode 2 corresponds to symmetric extension on the right and is used for computing y_3. Figure 3(b) illustrates the three modes. Figure 3(c) shows how the 3 modes can be implemented using simple combinational logic. □

In general, the number of modes in which a router operates is a function of K. As K increases, the overlap in the symmetrically extended signals of consecutive sequences increases. Consequently, the number of modes of a router increases and the complexity of the combinational logic increases. It is to be noted that the router design for the even-length and odd-length filters is the same. The actual implementations are however different, since the filters they interface with are of different sizes.

4. 2-D DWT ARCHITECTURE

In this section we first describe the 2-D DWT architecture for even-length filters and then briefly state the extensions that need to be done for odd-length filters. The proposed 2-D DWT architecture for even-length filters is an extension of the architecture in [4]. It consists of two parallel computation units which compute along the rows (Filters 1 and 2), two parallel computation units which compute along the columns (Filters 3 and 4), three storage units (Units 1, 2 and 3) and two routers (Routers A and B). Figure 4 gives the block diagram of this architecture. The inputs to Filters 1 and 2 come from either the input buffer or Storage Unit 3 via Router A, and the outputs are stored in Storage Units 1 and 2. The inputs to Filters 3 and 4 come from either Storage Unit 1 or Unit 2 via Router B, and the output of Filter 3 is stored in Storage Unit 3. The outputs of Filters 1 and 2 are stored in row major order and are read out along the columns while the output of Filter 3 is both stored as well as read out along the rows. We next describe the operations in each filter.

Filter 1 computes the low pass output (L) and the higher octave outputs $(LL)^aL$, $1 \le a < J$, along each row. Since the L outputs are computed every other cycle, the $(LL)^aL$ outputs can be computed in an interleaved manner. Filter 1 computes a total of $(N \times \frac{N}{2} + \frac{N}{2} \times \frac{N}{4} + \ldots \frac{N}{2^{J-1}} \times \frac{N}{2^J})$ $= \frac{2}{3}N^2(1-(\frac{1}{4})^J)$ and is active approximately 2/3 of the time.

Filter 2 computes the high pass output (H) and the higher octave outputs $(LL)^aH$ along the rows. Filter 2 also computes $\frac{2}{3}N^2(1-(\frac{1}{4})^J)$ and is active approximately 2/3 of the time.

Filter 3 computes the outputs $(LL)^aLL$, and $(LL)^aHL$ along the columns, $0 \le a < J$. It first computes the outputs $(LL)^aLL$ and N cycles later computes the outputs $(LL)^aHL$. The outputs $(LL)^aLL$ are read into Storage Unit 3. Filter 3 computes a total of $2(\frac{N}{2} \times \frac{N}{2} + \frac{N}{4} \times \frac{N}{4} + \ldots + \frac{N}{2^J} \times \frac{N}{2^J}) = \frac{2}{3}N^2(1-(\frac{1}{4})^J)$ and is active approximately 2/3 of the time.

Filter 4 computes the outputs $(LL)^aLH$, and $(LL)^aHH$ along the columns, $0 \le a < J$. It first computes the outputs of $(LL)^aLH$ and then (N cycles later) computes the outputs of $(LL)^aHH$. Filter 4 also computes a total of $\frac{2}{3}N^2(1-(\frac{1}{4})^J)$ and is active approximately 2/3 of the time.

4.1. Schedule

The scheduling algorithm is very similar to that of zero padded images described in [4]. The main difference here too is that the computations of the first output can start only after $K/2+1$ inputs have been read in. The times at which the different outputs are computed for even-length filters is as follows.

$L(i,j)$ and $H(i,j)$ are computed at cycle $Ni+2j+T+K/2$, $0 \le i < N$, $0 \le j < N/2$.

For $1 \le a \le J$, $(LL)^a(i,j)$ is computed at cycle $2^aNi+2^aj+2aT+K/2N\sum_{i=0}^{a-1}2^i+K/2\sum_{i=0}^{a-1}2^i+\sum_{i=1}^{a-1}\beta_i+\sum_{i=1}^{a-1}\gamma_i$, $0 \le i,j < N/2^a$. $(LL)^{a-1}LH(i,j)$ is computed at the same time as $(LL)^a(i,j)$. $(LL)^{a-1}HL(i,j)$ and $(LL)^{a-1}HH(i,j)$ are computed N cycles later.

$(LL)^aL(i,j)$ and $(LL)^aH(i,j)$ are computed at cycle $2^aNi+2^{a+1}j+(2a+1)T+K/2N\sum_{i=0}^{a-1}2^i+K/2\sum_{i=0}^{a}2^i+\sum_{i=1}^{a}\beta_i+\sum_{i=1}^{a-1}\gamma_i$, $0 \le i < N/2^a$, $0 \le j < N/2^{a+1}$.

The β and the γ values are chosen so that there is no scheduling conflict.

4.2. Router Design

Router A is required to support the pipelined execution of the row computations in Filters 1 and 2. Similarly, Router B is required to support the pipelined execution of the column computations in Filters 3 and 4. Both the routers are very similar to that described in Section 3. While the combinational logic of the two routers is the same, the control signals that dictate the mode of the routers are different. The control signals are a function of the exact schedule and have to be determined by the master controller. For instance, for $N \times N = 64 \times 64$, $T = 2$, $K = 4$, $J = 2$, Router A is (i) in Mode 0 at times $64i+2$, $0 \leq i < 64$ and $128j+139$, $0 \leq j < 32$, (ii) in Mode 2 at times $64i + 64$, $0 \leq i < 64$ and $128j + 199$, $0 \leq j < 32$, and (iii) in Mode 1 at all other times.

4.3. Storage Unit

In this section, we describe the details of the storage unit, namely, the sizes of each of the units and the way data can be organized in the units.

Storage Unit Size: Storage Unit 1 stores K rows of $N/2$ L outputs or $KN/2$ L outputs, $KN/8$ LL outputs, and so on. Thus the total size of Storage Unit 1 is $\approx \frac{2}{3}KN(1 - (\frac{1}{4})^J)$. Storage Unit 2 is larger that Unit 1 since the lifetimes of the variables stored in Unit 2 are larger. Recall that while the $(LL)^a L$ and the $(LL)^a H$ outputs are computed at the same time, the $(LL)^a H$ outputs are consumed N cycles after the $(LL)^a L$ outputs. As a result, Storage Unit 2 has to store K or more rows (depending on T, β, γ) for each of the J octaves. The size of Storage Unit 2 is $\approx \frac{2}{3}(K+1)N(1 - (\frac{1}{4})^J)$, if $K + 1$ rows are stored for each of the J octaves. Storage Unit 3 is comparatively small, since the lifetime of the $(LL)^a$ outputs is very small. The size of Storage Unit 3 is $\approx (K + 1)J$, if $K + 1$ elements are stored for each of the J octaves.

Data Organization: The data in the storage units can be organized in many different ways – in shift register arrays, monolithic memories, multiple memory banks, etc. Since multiple read/write accesses are required, shift register arrays and multiple memory banks are more suitable.

In the shift register array organization, Storage Unit 1 consists of J subarrays, where the ith subarray is of size $K \times \frac{N}{2^{2i+1}}$, $0 \leq i < J$. Similarly, Storage Unit 2 consists of J subarrays, where the ith subarray is of size $(K+1) \times \frac{N}{2^{2i+1}}$.

In the memory bank structure, each of the storage units can be organized into K or $K + 1$ memory banks. For instance, Storage Unit 1 can be organized into K memory banks with $(\frac{N}{2} + \frac{N}{8} + \ldots + \frac{N}{2^{J+1}}) = \frac{2}{3}N(1-(\frac{1}{4})^J)$ elements in each bank. Similarly, Storage Unit 2 can be organized into $K + 1$ memory banks with $\frac{2}{3}N(1 - (\frac{1}{4})^J)$ elements in each bank. In Storage Unit 1, the $(LL)^a L$ outputs are written into one memory bank at a time and are read out from all K banks at the same time. The multiple memory bank structure is area efficient. However, it comes at the expense of an increase in the router complexity. Figure 5 describes the router for the case when $N = 8$ and $K = 4$. Note that since the data is stored in the memory banks in the cyclic mode, the router complexity is more compared to the router described in Figure 3.

4.4. Architecture for odd-length filters

The architecture for odd-length filters is very similar to that for even-length filters for both 1-D and 2-D DWT. The main difference is that a router is required for each of the filters. This is because for odd-length filters, the high pass and the low pass filters are of different lengths, and thus the same router cannot be used to interface with both the filters. Furthermore, since the time that it takes to compute the

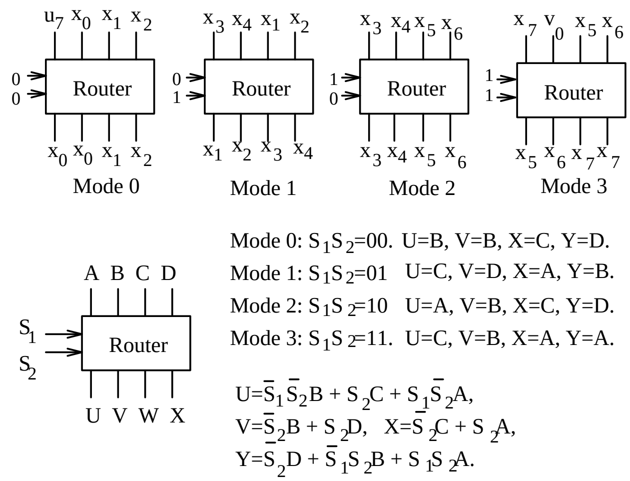

Figure 5. Router design when the storage unit is organized into memory banks for $N = 8$ and $K = 4$.

high pass and the low pass filter outputs is different, the storage unit sizes are different (since the sizes are a function of the lifetime of the stored variables).

ACKNOWLEDGEMENTS

The author would like to thank Prof. Lina Karam for many helpful discussions.

REFERENCES

[1] K. K. Parhi and T. Nishitani. VLSI architectures for discrete wavelet transforms. *IEEE Trans on VLSI Systems*, 1:191–202, 1993.

[2] M. Vishwanath, R.M. Owens, and M.J. Irwin. VLSI architectures for the discrete wavelet transform. *IEEE Trans. Circuits and Systems*, 42:305–316, May 1995.

[3] C. Chakrabarti and M. Vishwanath. Efficient realizations of the discrete and continuous wavelet transforms: from single chip implementations to SIMD parallel computers. *IEEE Trans. on Signal Processing*, 43:759–771, March 1995.

[4] C. Chakrabarti and C. Mumford. Efficient realizations of encoders and decoders based on the 2-D Discrete Wavelet Transform. to be published in the *IEEE Trans on VLSI Systems*, 1999.

[5] J. Fridman and E.S. Manolakos. Discrete Wavelet Transform: Data Dependence Analysis and Synthesis of Distributed Memory and Control Array Architectures. *IEEE Trans on Signal Processing*, vol4, no.5, May 1997.

[6] M. J. T. Smith and S. Eddins. Analysis/Synthesis techniques for subband image coding. *IEEE Trans on Acoustics, Speech and Signal Processing*, 1446–1456, August 1990.

[7] S. A. Matrucci. Non-expansive perfectly reconstructing analysis/synthesis filter banks. Georgia Institute of Technology, March 1991.