Efficient realizations of encoders and decoders based on the 2-D Discrete Wavelet Transform

Chaitali Chakrabarti and Clint Mumford

Abstract—
In this paper we present architectures and scheduling algorithms for encoders and decoders that are based on the 2-D Discrete Wavelet Transform. We consider the design of encoders and decoders individually, as well as in an integrated encoder-decoder system. We propose architectures ranging from SIMD processor arrays to folded architectures that are suitable for single chip implementations. The scheduling algorithms for the folded architectures range from those that try to minimize the latency to those that try to minimize the storage and keep the data flow regular. We include a comparison of the performance of these algorithms to aid the designer in choosing one that is best suited for a specific application.

Keywords— **Discrete Wavelet Transform architectures, folded implementation, SIMD array of processors, scheduling algorithms**

I. 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. This wide range of applications makes a study of their implementation in VLSI very important.

The wavelet transform can be viewed as a decomposition of a signal in the time-scale plane [4], [11], [12]. 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 realizations of the 2-dimensional (2-D) Discrete Wavelet Transform (DWT), where the signal as well as the time and scale parameters are discrete. We consider the implementation of 2-D DWT based analysis and synthesis filters in a video encoder-decoder system. We refer to the analysis filter as 'encoder' and the synthesis filter as 'decoder'. Further, we assume that the image is of size $N \times N$, and that the high pass and low pass filters (in the filter bank implementation) are each of size K.

There now exists a number of architectures for computing the 1-D DWT [1], [2], [3], [6], [8], [9], [14], [15]. The architectures for 1-D DWT can, in principle, be extended to architectures for separable 2-D DWT. This is because the separable 2-D DWT can be computed by the row-column method i.e., compute 1-D DWT on rows followed by 1-D DWT on columns (in contrast, the non-separable 2-D DWT computes on the 2-D data directly). Since minimizing latency and storage requirements are important in most applications, mapping 1-D DWT architectures to 2-D DWT architectures is not trivial. Lewis and Knowles [10] were the first to propose an architecture for the encoder based on the 2-D DWT. Their architecture is tuned to the Daubechies 4-tap filters and as a result does not work efficiently for other wavelets. Chakrabarti and Vishwanath [2] have proposed a scalable architecture for the encoder based on the non-separable 2-D DWT. Their architecture consists of 2 parallel computation units [1] of size K^2 and a storage unit of size $\approx NK$. Vishwanath, Owens and Irwin [14] have proposed an architecture for separable 2-D DWT which consists of 2 systolic arrays of size K, 2 parallel computation units of size K and a storage unit of size $\approx N(2K+J)$. The only drawback of [14] is that two rows of inputs are fed into the two systolic arrays every 2 cycles, and as a result an additional data converter is required to convert the raster scan input (1 per cycle) into 2 per 2 cycles output. Recently, Denk and Parhi [5] have proposed an encoder architecture based on the 2-D DWT which uses lapped block processing techniques to exploit the overlap in the input data blocks. The architecture uses minimum number of registers and the forward-backward allocation scheme to allocate the variables. Unfortunately, this comes at the expense of scalability and complex routing.

To date, we are aware of only one decoder architecture based on the inverse DWT (IDWT) [7]. The architecture was designed by Gordon, Meng and Chaddha for real-time video decompression. It consists of two filters (horizontal and vertical) each of size 4, and two storage units to store the intermediate outputs generated by the two filters. The scheduling of the subband levels is interleaved such that computations on a complete line of data from the four subbands at a given level is done before switching to another level. Since during the computation at a given level, the computations of other levels are not interleaved, such a scheduling scheme results in large latency.

In this paper we propose architectures and scheduling algorithms for encoders and decoders that are based on 2-D DWT. This is an extension of our earlier work that was presented in [13]. Our motivation is that while some applications require the latency or the computation time to be low (as in video conferencing) others require the memory size or the area to be low. Consequently, it is important to design architectures and algorithms to support this large spectrum of applications. In this paper we first describe implementation of the 2-D DWT and 2-D IDWT algorithms on a SIMD array of $N \times N$ processors. Such an implementation requires minimum computation time and can be supported by a parallel computer environment. Then we describe the folded implementation where in each cycle

C. Chakrabarti is with the Dept of Electrical Engg, Arizona State University. email:chaitali@asu.edu

[1] A parallel computation unit of size M consists of M multipliers and a tree of adders to add the M products.

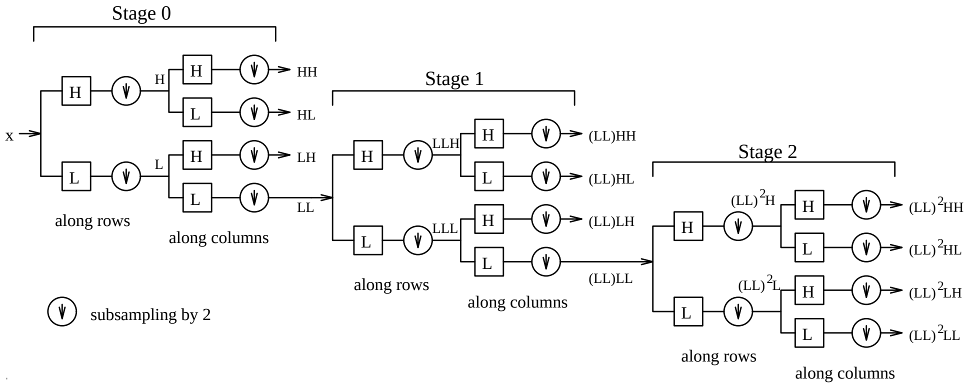

Fig. 1. The 2-D DWT filter bank structure for $J = 3$ stages.

only one data is input and one data is output making it suitable for single chip implementations in VLSI. We propose scheduling algorithms for the folded implementation of the encoders and decoders that try to (i) minimize the latency and/or (ii) try to minimize the storage and keep the data flow regular. We compare the performance of the scheduling algorithms with respect to latency, storage size and control unit complexity. We also consider the design of an integrated encoder-decoder system (where the decoder has to wait for the encoder outputs to be available). We compare the latency and the storage required in the design of such a system for the different scheduling algorithms. The aim of this performance analysis is to aid the designer in choosing an algorithm that is suitable for a specific application.

The rest of the paper is organized as follows. In Section 2 we briefly describe the subband structure of the encoder and decoder. In Section 3 we describe the SIMD implementation, the folded implementation and two scheduling algorithms for the encoder. Then in Section 4, we describe the SIMD implementation, the folded implementation and two scheduling algorithms for the decoder. We conclude the paper in Section 5.

II. Preliminaries

The encoder and the decoder can be implemented by the filter bank structures shown in Figures 1 and 2. We refer to the transform that is used in the encoder as DWT and the transform that is used in the decoder as IDWT. In Figure 1, 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.

In this paper we assume that (i) the image size is $N \times N$, (ii) the number of stages is J, (iii) the high pass and the low pass filters are of size K and have a latency of T cycles. We also assume that the borders have been padded with zeroes. We refer to the latency of the encoder as the forward-latency, the latency of the decoder as the reconstruction-latency, and that of the encoder-decoder system as the system-latency.

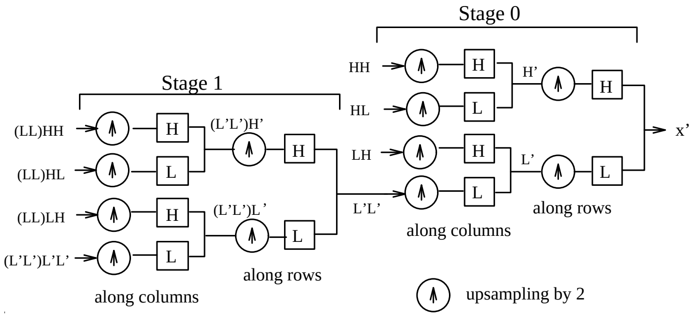

Fig. 2. The 2-D IDWT filter bank structure for $J = 2$ stages.

In the 2-D DWT filter bank structure, we denote the the high pass and the low pass outputs after the $(a+1)$th stage computations along the row by $(LL)^aH$ and $(LL)^aL$. The size of the $(LL)^aH$ and $(LL)^aL$ output arrays is $\frac{N}{2^a} \times \frac{N}{2^{a+1}}$, $0 \leq a < J$. $(LL)^aHH$ and $(LL)^aHL$ are the high pass and low pass outputs after computing along the columns of $(LL)^aH$, while $(LL)^aLH$ and $(LL)^aLL$ are the high pass and low pass outputs after computing along the columns of $(LL)^aL$. The size of the $(LL)^aHH$, $(LL)^aHL$, $(LL)^aLH$ and $(LL)^aLL$ output arrays is $\frac{N}{2^{a+1}} \times \frac{N}{2^{a+1}}$, $0 \leq a < J-1$.

In the 2-D IDWT filter bank structure, column computations are used to reconstruct $(L'L')^aH'$ from $(L'L')^aH'H'$ and $(L'L')^aH'L'$. Similarly, column computations are used to reconstruct $(L'L')^aL'$ from $(L'L')^aL'H'$ and $(L'L')^aL'L'$. The size of the $(L'L')^aH'$ and $(L'L')^aL'$ arrays is $\frac{N}{2^a} \times \frac{N}{2^{a+1}}$, $0 \leq a < J$. Row computations are used to reconstruct $(L'L')^a$ outputs from $(L'L')^aH'$ and $(L'L')^aL'$. The size of the $(L'L')^a$ outputs is $\frac{N}{2^a} \times \frac{N}{2^a}$, $0 \leq a < J$. Note that $(L'L')^0$ is equivalent to x', the output of the filter bank structure.

III. 2-D DWT Implementations

A. SIMD implementation

The subband structure for 2-D DWT (refer to Figure 1) for $N \times N$ data can be mapped onto a SIMD array of $N \times N$ processors. Each processor contains a reconfigurable switch such that if the switch is set to 1, the data passes through it with negligible delay. We refer to the processors which do multiply-add or which pass data to a neighboring processor with a delay of one cycle as 'active' processors. The reconfigurable switches can be used to reconfigure the processor array to a smaller array of active processors. Since the configuration patterns are known apriori, they can be stored in each processor.

In the proposed scheme, the row computations precede the column computations in each stage (see Figure 1). For the ath stage row computations, $0 \leq a < J$, the array is reconfigured to form $\frac{N}{2^a}$ row arrays each of size $\frac{N}{2^a}$ with processors $P(2^a i, 2^a j)$ active, $0 \leq i < \frac{N}{2^a}$, $0 \leq j < \frac{N}{2^a}$. For the ath stage column computations, the array is reconfigured to form $\frac{N}{2^{a+1}}$ column arrays each of size $\frac{N}{2^a}$ with processors $P(2^a i, 2^{a+1} j)$ active, $0 \leq i < \frac{N}{2^a}$, $0 \leq j < \frac{N}{2^{a+1}}$. After the ath stage computations, all the outputs are stored in processors $P(2^{a+1} i, 2^{a+1} j)$, $0 \leq i, j < \frac{N}{2^{a+1}}$. In this scheme, all the outputs of a particular stage are computed at the same time. The high pass and the low pass filter coefficients are broadcast to each processor. Each processor computes the products of the data elements and the filter coefficients, and updates the partial results of the low pass and the high pass outputs. The data element is then sent to the processor's active west neighbor during row computations and to the processors's active north neighbor during column computations. If each processor has only one multiplier, then for each stage, this scheme takes 2K cycles for the row computation and 4K cycles for the column computation.

We now propose a more efficient scheme where the column computation time can be reduced to 2K cycles. This is achieved by increasing the number of processors that are active. The procedure is as follows. After the ath stage row computation, the low pass outputs are stored in processors $P(2^a i, 2^{a+1} j)$ and the high pass outputs are sent to processors $P(2^a i, 2^{a+1} j + 2^a)$, $0 \leq i < \frac{N}{2^a}$ and $0 \leq j < \frac{N}{2^{a+1}}$. As a result processors $P(2^a i, 2^a j)$, $0 \leq i, j < \frac{N}{2^a}$ can now participate in the ath stage column computations. After the column computations, the low pass outputs are stored in processors $P(2^{a+1} i, 2^a j)$ and the high pass outputs are sent to processors $P(2^{a+1} i + 2^a, 2^a j)$, $0 \leq i < \frac{N}{2^{a+1}}$ and $0 \leq j < \frac{N}{2^a}$. The data in processors $P(2^{a+1} i, 2^{a+1} j)$ are now ready for the $(a+1)$th stage row computation, $0 \leq i, j < \frac{N}{2^{a+1}}$. At the end of this procedure, the $(LL)^J$ outputs are stored in processors $P(2^J i, 2^J j)$, $0 \leq i, j < \frac{N}{2^J}$. The $(LL)^a HL$ outputs are stored in processors $P(2^{a+1} i, 2^{a+1} j + 2^a)$, the $(LL)^a LH$ outputs are stored in processors $P(2^{a+1} i + 2^a, 2^{a+1} j)$, and the $(LL)^a HH$ outputs are stored in processors $P(2^{a+1} i + 2^a, 2^{a+1} j + 2^a)$, $0 \leq i, j < \frac{N}{2^{a+1}}$. The overall computation time of this scheme is $J(4K+2)$ cycles. Figure 3 describes the scheme in an example where the data is of size 8×8.

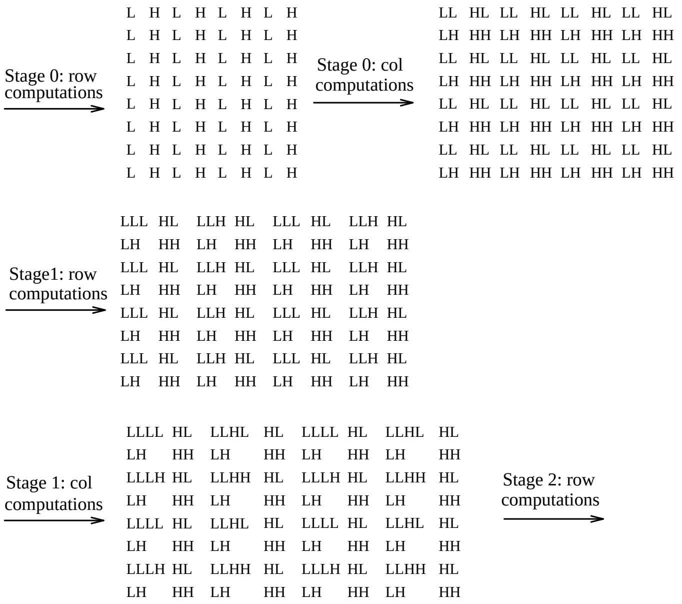

Fig. 3. SIMD array computation for 2-D DWT when the data array is of size 8×8.

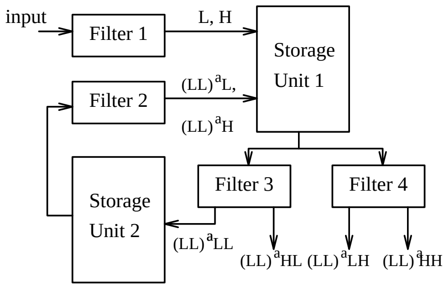

Fig. 4. Block diagram of Architecture 1 for 2-D DWT

B. Architecture 1: folded architecture

The SIMD array architecture described above can be folded onto an architecture which consists of two parallel computation units which compute along the rows, two parallel computation units which compute along the columns, and two storage units. Figure 4 gives the block diagram of this architecture. Here Filters 1 and 2 are the two parallel computation units which compute along the rows and Filters 3 and 4 are the parallel computation units which compute along the columns. The outputs of Filters 1 and 2 are stored in Storage Unit 1 in row major order and are read out along the columns by Filters 3 and 4. Similarly, the outputs of Filters 3 and 4 are stored in Storage Unit 2 along the columns and are read out along the rows by Filter 2. We next describe the operations in each filter.

Filter 1 computes the low pass and the high pass outputs (L and H) along each row. Since the low pass outputs are computed every other cycle, the low pass and the high pass outputs can be computed in an interleaved fashion. This filter computes $2(N * \frac{N}{2})$ outputs and is active all the time.

Filter 2 computes the outputs $(LL)^a L$ and $(LL)^a H$ along the rows, $1 \leq a < J$. It first computes the outputs $(LL)^a L$ and N cycles later computes the outputs $(LL)^a H$. This filter reads the same data from Storage Unit 2 twice, since the inputs for the computation of $(LL)^a L$ and $(LL)^a H$ are the same. Filter 2 computes a total of $2(\frac{N}{2} * \frac{N}{4} + \frac{N}{4} * \frac{N}{8} + \ldots) \approx \frac{N^2}{2}$ outputs and is active approximately 1/2 of the time.

Filter 3 computes the outputs $(LL)^a LL$, and $(LL)^a HL$ along the columns, $0 \leq a < J$. It first computes the outputs $(LL)^a LL$ and N cycles later computes the outputs $(LL)^a HL$. The outputs $(LL)^a LL$ are read into Storage

Unit 2. It computes a total of $2(\frac{N}{2} * \frac{N}{2} + \frac{N}{4} * \frac{N}{4} + \ldots) \approx \frac{2}{3}N^2$ and is active approximately 2/3 of the time.
Filter 4 computes the outputs $(LL)^aLH$, and $(LL)^aHH$ along the columns, $0 \leq a < J$. It first computes the outputs of $(LL)^aLH$ and then (N cycles later) computes the outputs of $(LL)^aHH$. This filter is also active approximately 2/3 of the time.

In the next subsection we describe two scheduling algorithms for this architecture. Since Filter 2 works in a lock step manner with Filters 3 and 4, if we determine a no-conflict scheduling scheme for Filters 3 and 4, the scheduling scheme for Filter 2 will automatically be conflict-free.

C. Scheduling Algorithms

In this section we describe two algorithms for scheduling the operations in Architecture 1. While Algorithm A tries to reduce the complexity of the control unit (at the expense of increase in forward-latency), Algorithm B tries to minimize the forward-latency.

The scheduling procedure that is followed by the two algorithms is described below. The two algorithms (A and B) differ in the way the $(LL)^a$ outputs get scheduled.

1. Schedule $L_{i,j}$ at time $T + Ni + 2j$, $0 \leq i < N$ and $0 \leq j < \frac{N}{2}$.
   Schedule $H_{i,j}$ 1 cycle after $L_{i,j}$.
2. for $a = 1$ to $J-1$ schedule $(LL)^a_{i,j}$, $0 \leq i,j < \frac{N}{2^a}$.
   Schedule $(LL)^{a-1}LH_{i,j}$ at the same time as $(LL)^a_{i,j}$.
3. for $a = 1$ to $J-1$, schedule $(LL)^{a-1}HL_{i,j}$ and $(LL)^{a-1}HH_{i,j}$ N cycles after $(LL)^a_{i,j}$, $0 \leq i,j < \frac{N}{2^a}$.
4. for $a = 1$ to $J-1$, schedule $(LL)^aL_{i,j}$ T cycles after $(LL)^a_{i,2j}$, $0 \leq i < \frac{N}{2^a}$, $0 \leq j < \frac{N}{2^{a+1}}$.
   Schedule $(LL)^aH_{i,j}$ N cycles after $(LL)^aL_{i,j}$.

C.1 Algorithm A

The primary goal of this scheduling algorithm is to minimize the size of the storage unit and reduce the complexity of the control unit. This algorithm gives the lower stage outputs a priority higher than the higher stage outputs. Moreover, the outputs are scheduled in a regular manner, i.e., $(LL)^a_{i,j}$ is scheduled 2^a cycles after $(LL)^a_{i,j-1}$. We summarize the scheduling of $(LL)^a_{i,j}$ below. The key step is determining a conflict-free schedule for $(LL)^a_{0,0}$, since this guarantees that there will be no conflicts for all other outputs of that stage.

---

1. Schedule $LL_{i,j}$ T cycles after $L_{2i,j}$, $0 \leq i < \frac{N}{2}$, $0 \leq j < N/2$.
2. for $a = 2$ to $J-1$ schedule $(LL)^a_{i,j}$, $0 \leq i,j < \frac{N}{2^a}$.
   Let $(LL)^a_{0,0}$ be scheduled $2T + D(a)$ cycles after $(LL)^{a-1}_{0,0}$. Find the minimum value of $D(a)$ such that there is no conflict with $(LL)^b_{i,j}$, $1 \leq b < a$. Once $D(a)$ is known, the remaining $(LL)^a_{i,j}$s are scheduled $2T + D(a)$ cycles after the corresponding $(LL)^{a-1}_{2i,2j}$s.

C.2 Algorithm B

The primary goal of this algorithm is to reduce the forward-latency at the expense of a mild increase in control unit complexity. This algorithm gives the highest priority to the scheduling of $(LL)^a_{0,0}$ outputs. It also gives the lower stage outputs a priority higher than the higher stage outputs. We summarize the key steps in the scheduling of $(LL)^a_{i,j}$ below. The procedure involves first finding an approximate schedule based on data dependencies, and then finding a time slot nearest to the approximate schedule where there would be no conflicts.

---

1. for $a = 1$ to $J-1$ schedule $(LL)^a_{0,0}$.
   $LL_{0,0}$ is scheduled at time 2T and $(LL)^a_{0,0}$ is scheduled 2T cycles after $(LL)^{a-1}_{0,0}$.
2. for $a = 1$ to $J-1$ schedule $(LL)^a_{i,j}$, $0 \leq i,j < \frac{N}{2^a}$, except $i = j = 0$.
   The approximate schedule of $LL_{i,j}$ is $2(Ni + j) + 2T$ and that of $(LL)^a_{i,j}$ is 2T cycles after $(LL)^{a-1}_{2i,2j}$. The approximate schedule is refined so that there are no conflicts with (i) $(LL)^b_{0,0}$, $a < b < J-1$ and (ii) $(LL)^b_{i,j}$, $1 \leq b < a$.

D. Storage Unit Size

Architecture 1 consists of two storage units (refer to Figure 4): Storage Unit 1 which stores the $(LL)^aL$ and the $(LL)^aH$ outputs and Storage Unit 2 which stores the $(LL)^a$ outputs. In this section we calculate the size of the two storage units. We first determine the approximate size and then show how lifetime analysis can be used to derive the exact size.
Storage Unit 1: The $(LL)^aL$ outputs are used for the computation of the $(LL)^aLL$ and the $(LL)^aLH$ outputs, $0 \leq a < J$. Since the lifetimes of adjacent $(LL)^aL$ columns significantly overlap [2], the memory required to store $(LL)^aL$ is approximately K times the number of columns in $(LL)^aL$, that is, $\approx KN/2^{a+1}$. The size of the memory that is required to store $(LL)^aL$ is the same as that of $(LL)^aH$ for $1 \leq a < J$. However the size of the memory that is required to store the H outputs is $N/2$ larger than the memory that is required to store the L outputs (which is $\approx KN/2$). The total size of Storage Unit 1 is $\approx N/2 + \sum_{a=0}^{J-1} 2KN/2^{a+1} = N/2 + 2KN(1-(\frac{1}{2})^J)$.

The exact size of the memory required to store the $(LL)^aL$ outputs can be calculated by first finding the life time (birth time and death time) of each output, and then finding the maximum number of live variables at any time. The death time of outputs $(LL)^aL_{2i,j}$ and $(LL)^aL_{2i-1,j}$ is (i) the time when $(LL)^aLL_{i+K/2-1,j}$ is scheduled or (ii) if $i + K/2 - 1 \geq \frac{N}{2^{a+1}} - 1$, the time when $(LL)^aLL_{N/2^{a+1}-1,j}$ is scheduled, $0 \leq i,j < \frac{N}{2^{a+1}}$. Once the death times are identified, the procedure of finding the maximum number of live variables is straight-forward.
Storage Unit 2: The $(LL)^a$ outputs that are stored in this memory is used for the computation of the $(LL)^aL$ and the $(LL)^aH$ outputs, $0 < a < J-1$. Since the $(LL)^aH$ outputs are computed N cycles after the $(LL)^aL$ outputs,

[2] The lifetimes of K columns do not actually overlap and so this estimate is larger than the actual value.

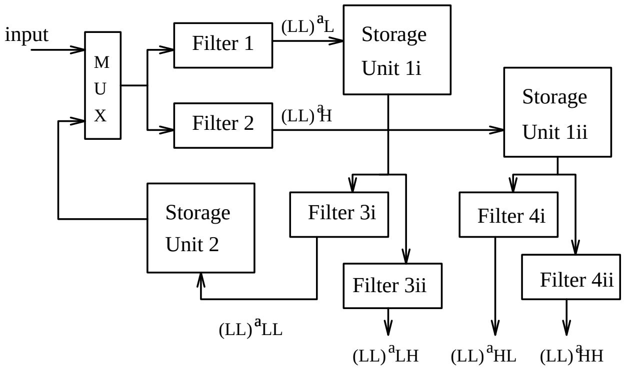

Fig. 5. Block diagram of Architecture 2 for 2-D DWT

Algorithm	T=3	T=4	T=5	T=6
A	95	103	107	115
B	88	96	104	112
modA	31	39	43	51
modB	24	32	40	48
minimum possible	24	32	40	48

Fig. 6. Comparison of the forward-latency of the encoder for different values of T when the data size is 64×64 and $J = 4$.

the $(LL)^aH$ outputs determine the size of this memory. The memory required to store $(LL)^a$ is $\approx N/2^a$, and thus the total size is $\approx \sum_{a=1}^{J-1} N/2^a = N(1-(\frac{1}{2})^{J-1})$.

The exact size of the memory required to store the $(LL)^a$ outputs is also calculated by lifetime analysis. The death time of outputs $(LL)^a_{i,2j}$ and $(LL)^a_{i,2j-1}$ is (i) the time when $(LL)^a H_{i,j+K/2-1}$ is scheduled or (ii) if $j + K/2 - 1 \geq \frac{N}{2^{a+1}} - 1$, the time when $(LL)^a H_{i,N/2^{a+1}-1}$ is scheduled, $0 \leq j < \frac{N}{2^{a+1}}$, $0 \leq i < \frac{N}{2^a}$. Since the lifetimes of the rows do not overlap, lifetime analysis is done on each row, and the maximum number of live variables in each row is used to determine the size of the corresponding storage.

E. Architecture 2: modified folded architecture

In Architecture 1 (see Figure 4), the $(LL)^aLH$ outputs are produced N cycles after the $(LL)^aLL$ outputs. Similarly the $(LL)^aHH$ outputs are produced N cycles after the $(LL)^aHL$ outputs. As a result, reconstruction (in the decoder) cannot start untill the $(LL)^aLH$ outputs and the $(LL)^aHH$ outputs have been computed. In applications where it is important to reduce the latency of the encoder-decoder system, a delay of N cycles may not be acceptable. In this section we propose a modification of Architecture 1 where the outputs $(LL)^aLL$, $(LL)^aLH$, $(LL)^aHL$ and $(LL)^aHH$ are produced at the same time. The resulting architecture, Architecture 2, has a latency that is N cycles lower than that of Architecture 1.

Figure 5 gives the block diagram of Architecture 2. Filters 1 and 2 are parallel computation units which compute along the rows, and Filters 3i, 3ii, 4i and 4ii are parallel computation units which compute along the columns. The parameter a lies in the range $0 \leq a < J-1$. Filter 1 computes the low pass outputs of the form $(LL)^aL$ and stores the outputs in row major order in Storage Unit 1i. Filter 2 computes the high pass outputs of the form $(LL)^aH$ and stores the outputs in row major order in Storage Unit 1ii. Filters 3i and 3ii read the data from Storage Unit 1i and compute the outputs $(LL)^aLL$ and $(LL)^aLH$. Similarly, Filters 4i and 4ii read the data from Storage Unit 1ii and compute the outputs $(LL)^aHL$ and $(LL)^aHH$. The $(LL)^a$ outputs of Filter 3i are stored in Storage Unit 2. Filters 1 and 2 compute on data that is either read from Storage Unit 2 or from the x input data stream. Since Filters 3i and 3ii work in a lock-step manner with Filter 1, and since Filters 4i and 4ii work in a lock-step manner with Filter 2, if we determine a conflict-free schedule for Filters 1 and 2, the scheduling scheme for Filters 3i, 3ii, 4i, and 4ii will be conflict-free. Moreover, it is sufficient to design a conflict-free schedule for only of the filters, say, Filter 1.

Scheduling procedure:

1. Schedule $L_{i,j}$ and $H_{i,j}$ at time $T+Ni+2j$, $0 \leq i < N$ and $0 \leq j < \frac{N}{2}$.
2. for $a = 1$ to $J-1$, schedule $(LL)^aL_{i,j}$, $0 \leq i < \frac{N}{2^a}$, $0 \leq j < \frac{N}{2^{a+1}}$. Schedule $(LL)^aH_{i,j}$ at the same time as $(LL)^aL_{i,j}$.
3. for $a = 1$ to $J-1$, schedule $(LL)^{a-1}LL_{i,j}$ T cycles after $(LL)^{a-1}L_{2i,j}$, $0 \leq i,j < \frac{N}{2^a}$. Schedule $(LL)^{a-1}LH_{i,j}$, $(LL)^{a-1}HL_{i,j}$ and $(LL)^{a-1}HH_{i,j}$ at the same time as $(LL)^{a-1}LL_{i,j}$.

The scheduling algorithms (A and B) that have been described in Section 3.3 to schedule the $(LL)^a$ outputs can be extended to schedule the $(LL)^aL$ outputs in Step 2 of the scheduling procedure. The extension is minor and we will not describe it here.

Storage Unit Size:

The storage unit size of Architecture 2 is estimated in a way similar to that of Architecture 1. Storage Unit 1i and 1ii are each of size $\approx KN(1-(\frac{1}{2})^J)$ and Storage Unit 2 is of size $N(1-(\frac{1}{2})^{J-1})$.

F. Comparison

In this section, we compare the performance of Algorithms A and B with respect to forward-latency, storage size and simplicity of the control unit. We refer to the scheduling algorithms of Architecture 2 as modA and modB.

Figure 6 compares the forward-latency of the encoder for different values of T. Here forward-latency is defined as the difference in the time between when the first x-value is input and when the first $(LL)^{J-1}HL$ value is output. Note that Architecture 2 has a forward-latency that is N cycles less than that of Architecture 1.

The control circuitry to implement the schedule of Algorithm A consists of a counter and very simple logic circuitry. Since the schedule of Algorithm B is not as regular as Algorithm A, the control logic circuitry is not as simple. Figure 7 compares the control circuit complexity of the different implementations.

Algorithm	T=3	T=4	T=5	T=6
A	very low	very low	very low	very low
B	moderate	moderate	low	moderate
modA	very low	very low	very low	very low
modB	moderate	moderate	low	moderate

Fig. 7. Comparison of the control circuit complexity of the encoder for different values of T when the data size is 64×64 and $J = 4$.

Algorithm	K=4,T=3	K=4,T=4	K=6,T=3	K=6,T=4
A	432(376+56)	436(380+56)	658(602+56)	660(604+56)
B	432(376+56)	432(376+56)	656(600+56)	656(600+56)
approx	568(512+56)	568(512+56)	808(752+56)	808(752+56)
modA	400(344+56)	404(348+56)	624(568+56)	628(572+56)
modB	400(344+56)	404(348+56)	624(568+56)	628(572+56)
approx	536(480+56)	536(480+56)	776(720+56)	776(720+56)

Fig. 8. Comparison of the storage required in the encoder for different values of K and T when the data size is 64×64 and $J = 4$. Notation A(B+C): A is the total storage, B is the size of Storage Unit 1 and C is the size of Storage Unit 2.

Figure 8 compares the storage that is required in the encoder for different values of N, K and T. Note that Architecture 2 requires $N/2$ less storage compared to Architecture 1. The (approximate) storage sizes derived in Section 3.4 have also been added for comparison. While the actual size of Storage Unit 2 matches the estimated value, the actual size of Storage Unit 1 is a lot smaller (as is expected).

IV. 2-D IDWT implementation

A. SIMD array implementation

The subband structure for 2-D IDWT (refer to Figure 2) for $N \times N$ data can be mapped onto a SIMD array of $N \times N$ processors. As in the case of the processors in the 2-D DWT architecture, each processor contains a reconfigurable switch. The purpose of this switch is to enable the $N \times N$ processor array to be reconfigured into a smaller array of active processors.

The 2-D IDWT computations are done on data (2-D DWT outputs) that is distributed in the following way. If the number of stages is J, then the $(LL)^J$ outputs are stored in processors $P(2^J i, 2^J j)$, $0 \leq i,j < \frac{N}{2^J}$. For $0 \leq a < J$, the $(LL)^a HL$ outputs are stored in processors $P(2^{a+1}i, 2^{a+1}j + 2^a)$, the $(LL)^a LH$ outputs are stored in processors $P(2^{a+1}i + 2^a, 2^{a+1}j)$, and the $(LL)^a HH$ outputs are stored in processors $P(2^{a+1}i + 2^a, 2^{a+1}j + 2^a)$, $0 \leq i,j < \frac{N}{2^{a+1}}$. The IDWT computations begin with Stage $J-1$ (See Figure 2). In each stage, the column computations precede the row computations. For the ath stage column computations, the array is reconfigured to form $\frac{N}{2^a}$ column arrays each of size $\frac{N}{2^a}$. Out of these, only $\frac{N}{2^{a-1}}$ processors in each column array do the multiply accumulate operations. For the ath stage row computations, the array is reconfigured to form $\frac{N}{2^a}$ row arrays each of size $\frac{N}{2^a}$. Once again only $\frac{N}{2^{a-1}}$ processors in each row array do the multiply accumulate operations. The reconstructed input $x'_{i,j}$ is finally available in processor $P(i,j)$, $0 \leq i,j < N-1$.

In this scheme, the even numbered outputs, $\{z_{2j}\}$, and the odd numbered outputs, $\{z_{2j+1}\}$, are computed in an interleaved fashion. This is because the same input data is required for the computation of two consecutive (one even numbered and one odd numbered) outputs. Let g_0 through g_{K-1} be the low pass filter coefficients and let h_0 through h_{K-1} be the high pass filter coefficients. Then to compute the even numbered outputs, coefficients $\{g_{2i}, h_{2i}\}$ have to be broadcast and to compute the odd numbered outputs, coefficients $\{g_{2i+1}, h_{2i+1}\}$ have to be broadcast, $0 \leq i < \frac{K}{2}$. The sequence in which the coefficients are broadcast is as follows: $g_0, g_1, h_0, h_1, \ldots, g_{K-2}, g_{K-1}, h_{K-2}, h_{K-1}$. Each processor computes the product of the filter coefficient and the data element and updates the partial result. After the partial results of an even numbered ($(2j)$th) output and an odd numbered ($(2j+1)$th) output have been updated, the data element is sent to the processor's active north neighbor during column computations and to the processor's active west neighbor during row computations. After $2K$ cycles, the even numbered ($(2j)$th) and the odd numbered ($(2j+1)$th) output are in the same processor. The odd numbered output is then sent to the active south neighbor during column computations and to the active east neighbor during row computations. For instance, after the ath stage column computations, processors $P(2^{a+1}i, 2^a j)$ contain the outputs $(2i, j)$ and $(2i+1, j)$, $0 \leq i < \frac{N}{2^{a+1}}$, and $0 \leq j < \frac{N}{2^a}$. The $(2i+1, j)$th outputs are then sent to processors $P(2^{a+1}i + 2^a, 2^a j)$. The overall computation time of this scheme is $J(2K+1)$.

B. Architecture 3: folded architecture

The SIMD array architecture described in Section 4.1 can be folded onto an architecture that is suitable for single chip implementation. The proposed folded architecture, Architecture 3, 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), and two storage units as shown in Figure 9. The input is read in from the encoder memory in row major order by Filters 1 and 2, and the output x' is

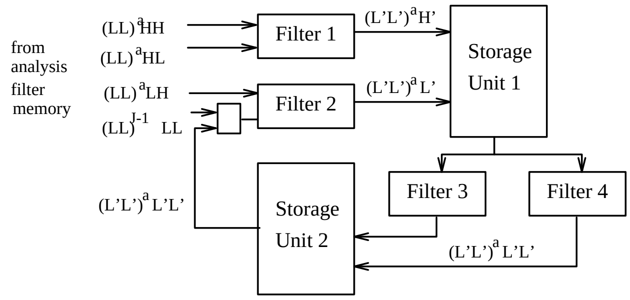

Fig. 9. Block diagram of Architecture 3 for 2-D IDWT

generated in row major order by Filters 3 and 4. Filters 1 and 2 compute along the columns and store the outputs in Storage Unit 1 in the row major order. Filters 3 and 4 read the data out of Storage Unit 1 along the rows and write the outputs in Storage Unit 2 in row major order. The specific operations in each filter are described next.

Filter 1 is a column filter which reads data $(LL)^a HH$ and $(LL)^a HL$ from the encoder memory and computes the outputs $(L'L')^a H'$, $J > a \geq 0$. It reads the input data along the columns but produces the outputs in a row major order. Since the same set of data is required for the computation of two rows of $(L'L')^a H'$, Filter 1 reads in the same data twice. The multipliers in Filter 1 are programmable since the same input data is used to produce two outputs, namely, $(L'L')^a H'_{2i,j}$ and $(L'L')^a H'_{2i+1,j}$, $0 \leq i < \frac{N}{2^{a+1}}$, at two different times. Filter 1 computes $(\frac{N}{2} * N + \frac{N}{4} * \frac{N}{2} + \ldots) \approx \frac{2}{3}N^2$ outputs and is active $\frac{2}{3}$ of the time.

Filter 2 is also a column filter which reads data $(LL)^a LH$ and $(LL)^{J-1}$ from the encoder memory and $(L'L')^a L'L'$ from Storage Unit 2, $J-1 > a \geq 0$. It reads the input data along the columns and computes the outputs $(L'L')^a L'$ in row major order. The multipliers in this filter are also programmable since the same input is required for the computation of two rows of outputs. Filter 2 also computes $\approx \frac{2}{3}N^2$ outputs and is active $\frac{2}{3}$ of the time.

Filters 3 and 4 are row filters that read $(L'L')^a H'$ and $(L'L')^a L'$ from Storage Unit 1 and compute $(L'L')^a$, $J > a \geq 0$. These two filters operate in a way different from Filters 1 and 2. Essentially both the filters operate on the same data at the same time and produce two consecutive outputs, $(L'L')^a_{i,2j}$ and $(L'L')^a_{i,2j+1}$, at the same time. Both filters 3 and 4 consist of a fixed set of K multipliers. The multipliers in Filter 3 consist of the odd indexed coefficients of both the high pass and low pass filters, while those in Filter 4 consist of the even indexed coefficients of both the high pass and the low pass filters. For instance, in an example with $L = 4$, if g_0 through g_3 are the coefficients of the high pass filter and if h_0 through h_3 are the coefficients of the low pass filter, Filter 3 consists of multipliers corresponding to g_1, g_3, h_1 and h_3, and Filter 4 consists of the multipliers corresponding to g_0, g_2, h_0 and h_2. The two outputs that are produced simultaneously by Filters 3 and 4 are then written into Storage Unit 2. Filters 3 and 4 compute $(N * N + \frac{N}{2} * \frac{N}{2} + \ldots) \approx \frac{4}{3}N^2$ outputs and are active $\frac{2}{3}$ of the time.

In the next section we describe two scheduling algorithms for this architecture. Filters 3 and 4 are scheduled T cycles after Filters 1 and 2. So if we determine a no-conflict scheduling scheme for Filters 1 and 2, the scheduling scheme for Filters 3 and 4 will also be conflict-free.

C. Scheduling Algorithms

In this section we describe two algorithms for scheduling the operations in Architecture 3. Both algorithms generate the x' outputs in a regular manner, i.e., two x' outputs are produced every two cycles. This feature is important since it allows the reconstructed outputs to be sent for further processing without undergoing buffering or data conversion (for I/O compatibility). While Algorithm C tries to minimize the size of the storage unit and reduce the complexity of the control unit, Algorithm D tries to reduce the reconstruction-latency at the expense of increasing the control unit complexity.

C.1 Algorithm C

The primary goal of this scheduling algorithm is to minimize the size of the storage unit and reduce the complexity of the control unit. This comes at the expense of an increase in the reconstruction-latency. To reduce the size of the storage unit, the outputs are scheduled in a regular manner. For instance, $(L'L')^a L'_{i,j}$ is scheduled 2^{a+1} cycles after $(L'L')^a L'_{i,j-1}$. The outputs are scheduled from the lower stages to the higher stages. This guarantees that in case of conflict, the lower stage outputs have a priority higher than the higher stage outputs. The key step in this algorithm is determining a conflict free schedule for the $(L'L')^a L'$ outputs, $1 \leq a < J$. Once this scheduling is determined, the schedule of the other outputs are determined in a straightforward manner. The final step involves delaying the entire schedule if there are any conflicts with the DWT schedule.

In the determination of a schedule for the $(L'L')^a L'$ outputs, we first determine $D(a)$, the delay by which the scheduling of all the $(L'L')^a L'$ outputs get postponed. The value of $D(a)$ is calculated so that there is no conflict between the schedule of $(L'L')^a L'$ and that of the other $(L'L')^b L'$ outputs, $0 \leq b < a$. Note that since $D(a)$ delays the whole schedule, the latency increases but the outputs are scheduled in a regular manner. To find $D(a)$, it is sufficient to consider only the 0th row of $(L'L')^a L'$ and $(L'L')^b L'$, $0 \leq b < a$.

The $(L'L')^a L'_{0,q}$ outputs, $0 \leq q < \frac{N}{2^{a+1}}$, are scheduled at time

$$-a2T - \sum_{j=1}^{a-1} D(j) - D(a) + 2^{a+1}q$$

To avoid conflict, $D(a)$ should be chosen such that for $0 \leq b < a$,

$$-a2T - \sum_{j=1}^{a-1} D(j) - D(a) + 2^{a+1}q \neq -b2T - \sum_{j=1}^{b} D(j) + 2^{b+1}m,$$

$0 \leq q < \frac{N}{2^{a+1}}$ and $0 \leq m < \frac{N}{2^{b+1}}$. In other words, for $0 \leq b < a$,

$$2^{b+1}(2^{a-b}q - m) \neq (a-b)2T + D(a) + \sum_{j=b+1}^{a-1} D(j)$$

The procedure to find the smallest value of $D(a)$ which satisfies the a disequations (corresponding to each value of b) is as follows: (i) for each disequation find $S(b, D(a))$, the values of $D(a)$ which ensure that the RHS is not a multiple of 2^{b+1}, and (ii) finding the smallest value of $D(a)$ which a member of $\bigcap_b S(b, D(a))$. We explain this procedure with the help of an example.

Let the latency of the computation unit be $T = 2$. Then $D(1)$ is the smallest integer that satisfies the disequation $2(2q - m) \neq 4 + D(1)$. Since $S(0, D(1)) = \{1, 3, 5, 7, \ldots\}$, $D(1) = 1$. Similarly $D(2)$ is the smallest integer that satisfies (i) $2(4q - m) \neq 9 + D(2)$ and (ii) $4(2q - m) \neq 4 + D(2)$. Since $S(0, D(2)) = \{0, 2, 4, 6, \ldots\}$ and $S(1, D(2)) = \{1, 2, 3, 5, \ldots\}$, $D(2) = 2$. Thus in this example if the 0th row of L' is scheduled at time $2q_1$, $0 \leq q_1 < N/2$, then the 0th row of $(L'L')L'$ is scheduled at time $-5 + 4q_2$, $0 \leq q_2 < N/4$ and the 0th row of $(L'L')^2L'$ is scheduled at time $-11 + 8q_3$, $0 \leq q_3 < N/8$.

---

1. Schedule $x'_{i,2j}$, $0 \leq i < N$ and $0 \leq j < \frac{N}{2}$.
 $x'_{i,2j}$ is scheduled at time $Ni + 2j$. $x'_{i,2j}$ and $x'_{i,2j+1}$ are scheduled at the same time.
2. Schedule $L'_{i,j}$, $0 \leq i < N$, and $0 \leq j < \frac{N}{2}$.
 $L'_{i,j}$ is scheduled T cycles before $x'_{i,2j}$. Schedule $H'_{i,j}$ at the same time as $L'_{i,j}$.
3. for $a = 1$ to $J - 1$, schedule $(L'L')^a L'_{i,j}$, $0 \leq i < \frac{N}{2^a}$, $0 \leq j < \frac{N}{2^{a+1}}$.
 Schedule $(L'L')^a H'_{i,j}$ at the same time as $(L'L')^a L'_{i,j}$.
4. for $a = 1$ to $J - 1$, schedule $(L'L')^a_{i,j}$, $0 \leq i, j < \frac{N}{2^a}$.
 Schedule $(L'L')^a_{i,2j}$ T cycles after $(L'L')^a L'_{i,j}$, $0 \leq i < \frac{N}{2^a}$, and $0 \leq j < \frac{N}{2^{a+1}}$. Schedule $(L'L')^a_{i,2j+1}$ at the same time as $(L'L')^a_{i,2j}$.
5. Check for conflicts with the DWT schedule.
 Check whether $(L'L')^{J-1}H'_{2i,j}$ is scheduled T cycles after $(LL)^{J-1}HL_{i,j}$, $0 \leq i, j < \frac{N}{2^J}$. If there is a conflict, then delay the schedule of all outputs appropriately.

C.2 Algorithm D

The primary goal of this algorithm is to reduce the reconstruction-latency and generate the x' outputs in a regular fashion. This comes at the expense of a mild increase in the complexity of the control unit. In this procedure low reconstruction-latency is achieved by scheduling the $(L'L')^a_{0,0}$ outputs first. Next, the x' outputs are scheduled so that they are produced in a regular manner. The remaining outputs are scheduled from the lower stage to the higher stages. Finally, the entire schedule is delayed if there are any conflicts with the schedule of the DWT module.

---

1. for $a = J - 1$ down to 0, schedule $(L'L')^a_{0,0}$ and $(L'L')^a L'_{0,0}$.
 Schedule $(L'L')^{J-1}L'_{0,0}$ T cycles after $(LL)^{J-1}LL0,0$. Schedule $(L'L')^{J-1}_{0,0}$ T cycles after $(L'L')^{J-1}L'_{0,0}$. Schedule $(L'L')^{a-1}L'_{0,0}$ $2T$ cycles after $(L'L')^a L'_{0,0}$, and schedule $(L'L')^{a-1}_{0,0}$ $2T$ cycles after $(L'L')^a_{0,0}$, $J - 1 > a \geq 1$.
2. Schedule $x'_{i,2j}$, $0 \leq i < N$ and $0 \leq j < \frac{N}{2}$.
 $x'_{0,0}$ is equivalent to $(L'L')^0_{0,0}$ and is scheduled in Step 1. $x'_{i,2j}$ is scheduled $Ni + 2j$ cycles after $x'_{0,0}$. $x'_{i,2j}$ and $x'_{i,2j+1}$ are scheduled at the same time.
3. Schedule $L'_{i,j}$ $0 \leq i < N$, and $0 \leq j < \frac{N}{2}$.
 $L'_{i,j}$ is scheduled T cycles before $x_{i,2j}$. Schedule $H'_{i,j}$ at the same time as $L'_{i,j}$.
4. for $a = 1$ to $J - 1$ schedule $(L'L')^a L'_{i,j}$, $0 \leq i < \frac{N}{2^a}$, $0 \leq j < \frac{N}{2^{a+1}}$, except $i = j = 0$. Schedule $(L'L')^a H'_{i,j}$ at the same time as $(L'L')^a L'_{i,j}$.
 The approximate schedule of $(L'L')^a L'_{i,j}$ is $2T$ cycles before $(L'L')^{a-1}L'_{2i,2j}$. The approximate schedule is refined so that there are no conflicts with $(L'L')^b L'$, $0 \leq b < a$.
5. for $a = 1$ to $J - 1$, schedule $(L'L')^a_{i,j}$, $0 \leq i, j < \frac{N}{2^a}$.
 Schedule $(L'L')^a_{i,2j}$ T cycles after $(L'L')^a L'_{i,j}$, $0 \leq i < \frac{N}{2^a}$, and $0 \leq j < \frac{N}{2^{a+1}}$. Schedule $(L'L')^a_{i,2j+1}$ at the same time as $(L'L')^a_{i,2j}$.
6. Check for conflicts with the DWT schedule.
 Check whether $(L'L')^{J-1}H'_{2i,j}$ is scheduled T cycles after $(LL)^{J-1}HL_{i,j}$, $0 \leq i, j < \frac{N}{2^J}$. If there is a conflict, then delay the schedule of all outputs appropriately.

D. Storage Unit Size

Architecture 3 consists of two storage units (refer to Figure 9): Storage Unit 1 stores the $(L'L')^a H'$ and the $(L'L')^a L'$ outputs and Storage Unit 2 stores the $(L'L')^a$ outputs. In this section we first determine the approximate size of the storage units and then show how lifetime analysis can be used to derive the exact size.

Storage Unit 1: The $(L'L')^a H'$ and the $(L'L')^a L'$ outputs that are stored in this unit are used for the computation of the $(L'L')^a$ outputs, $0 \leq a < J - 1$. Since the lifetimes of adjacent rows do not overlap, it is sufficient to estimate the size of the memory required for one row. Accordingly, the size of the $(L'L')^a H'$ memory is $\approx K/2$ and that of the $(L'L')^a L'$ memory is $\approx K/2$. The total size is thus $\approx \sum_{a=0}^{J-1} K = KJ$.

The exact size of the memory is calculated by doing a lifetime analysis of the $(L'L')^a L'$ and the $(L'L')^a H'$ outputs. Since the memory required to store the $(L'L')^a L'$ and the $(L'L')^a H'$ outputs is the same, we consider only the $(L'L')^a L'$ outputs. The death time of $(L'L')^a L'_{i,j}$ is (i) the time when $(L'L')^a_{i,2j+K-1}$ is scheduled or (ii) if $2j + K - 1 \geq \frac{N}{2^a} - 1$, the time when $(L'L')^a_{i,N/2^a-1}$ is scheduled, $0 \leq j < \frac{N}{2^{a+1}}$, $0 \leq i < \frac{N}{2^a}$. Since the lifetimes of the rows do not overlap, lifetime analysis is done on each row, and the maximum number of live variables in each row is used to determine the size of the corresponding storage.

Storage Unit 2: The $(L'L')^a$ outputs stored in this memory are used for the computation of the $(L'L')^{a-1}L'$ outputs, $1 \leq a < J$. Note that the outputs needed for the computation of $(L'L')^{a-1}H'$ outputs are read out from the channel memory and its size is not considered here. Since the lifetimes of adjacent $(L'L')^a$ columns overlap significantly, the memory required to store the $(L'L')^a$ outputs is approximately $K/2$ times the number of columns, that is, $\approx KN/2^{a+1}$. The total size of Storage Unit 2 is then $\approx \sum_{a=1}^{J-1} KN/2^{a+1} = KN/2(1 - (\frac{1}{2})^{J-1})$.

The exact size can be calculated by analysing the lifetimes of the $(LL)^a$ outputs. The death time of output $(L'L')^a i, j$ is (i) the time when $(L'L')^{a-1}L'_{2i+K-1,j}$ is

Algorithm	T=3	T=4	T=5	T=6
C	31	39	43	51
D	21	28	35	42
minimum possible	21	28	35	42

Fig. 10. Comparison of the reconstruction-latency for different values of T when the data size is 64×64 and $J = 4$.

Algorithm	T=3	T=4	T=5	T=6
C	very low	very low	very low	very low
D	moderate	low	low	low

Fig. 11. Comparison of the control circuit complexity of the decoder for different values of T when the data size is 64×64 and $J = 4$.

scheduled or (ii) if $2i + K - 1 \geq \frac{N}{2^{a-1}} - 1$, the time when $(L'L')^{a-1}L'_{N/2^{a-1}-1,j}$ is scheduled, $0 \leq i < \frac{N}{2^a}$. The death times are necessary for determining the maximum number of live variables at any time.

E. Comparisons

In this section, we compare the performance of Algorithms C and D with respect to reconstruction-latency, storage size and simplicity of the control unit. We also compare the system-latency and the storage requirements (including the inter-filter storage) of the encoder-decoder system for different scheduling algorithms.

Figure 10 compares the reconstruction-latency of the decoder for different values of T. Here reconstruction-latency is defined as the difference in the time between when the first $(LL)^J$ value is input and when the first x' value is output. While calculating the reconstruction-latency, we assume that all the encoder outputs are available at the time the decoder starts its computations. Note that Algorithm D requires significantly smaller reconstruction-latency compared to Algorithm C and would clearly be the choice in applications where low reconstruction latency is very important.

The control circuitry to implement the schedule of Algorithm C consists of a counter and very simple logic circuitry. Since the schedule of Algorithm D is not as regular as Algorithm C, the control logic circuitry is not as simple. Figure 11 compares the control circuit complexity for different values of T.

Figure 12 compares the storage that is required in the decoder for different values of N, K and T. The (approximate) storage sizes derived in Section 4.4 have also been added for comparison. Note that the estimated sizes match well with the actual sizes.

We next compare the system-latency and storage requirements of an encoder-decoder system. Figure 13 compares the system latencies for different implementations. Note that the system-latency is $\approx N$ cycles lower if Architecture 2 is used instead of Architecture 1 in the encoder.

Algorithm	K=4,T=3	K=4,T=4	K=6,T=3	K=6,T=4
A	432(376+56)	436(380+56)	658(602+56)	660(604+56)
B	432(376+56)	432(376+56)	656(600+56)	656(600+56)
approx	568(512+56)	568(512+56)	808(752+56)	808(752+56)
modA	400(344+56)	404(348+56)	624(568+56)	628(572+56)
modB	400(344+56)	404(348+56)	624(568+56)	628(572+56)
approx	536(480+56)	536(480+56)	776(720+56)	776(720+56)

Fig. 12. Comparison of the storage required in the decoder different values of K and T when $N \times N = 64 \times 64$ and $J = 4$. Notation A(B+C): A is the size of the total storage, B is the size of Storage Unit 1 and C is the size of Storage Unit 2.

Algorithm	32x32	64x64
A - C	94	126
modA - C	62	62
B - C	94	126
modB - C	62	62
A - D	95	127
modA - D	62	62
B - D	91	127
modB - D	62	62

Fig. 13. Comparison of the system-latency of the encoder-decoder system for different values of N when $J = 4$, $K = 4$ and $T = 3$.

An interesting point to note is that while Algorithm D had a lower reconstruction-latency compared to Algorithm C (see Figure 10), the performance of Algorithms C and D with respect to system-latency is not so different. This is because in the encoder-decoder system, the decoder module has to wait for the encoder module to finish the required computations, thereby increasing the system-latency.

In the design of an encoder-decoder system, the inter-filter storage requirements (that is storage between the encoder and the decoder) is an important issue. This is because the outputs produced by the encoder are not processed by the decoder immediately and hence provisions have to be made to store that data. Figure 14 compares

Algorithm	NxN=32x32			NxN=64x64		
	Intra-filter	Inter-filter	Total	Intra-filter	Inter-filter	Total
A - C	222+68	224	514	436+124	395	955
modA - C	204+68	217	489	404+124	368	896
B - C	220+68	229	517	432+124	397	953
modB - C	204+68	197	469	404+124	368	896
A - D	222+70	224	516	436+126	395	957
modA - D	204+70	217	491	404+126	368	898
B - D	220+70	224	514	432+126	397	955
modB - D	204+70	195	469	404+126	368	898

Fig. 14. Comparison of the storage required in an encoder-decoder system for different values of N when $J = 4$, $K = 4$ and $T = 3$.

the inter-filter storage as well as the intra-filter storage for different values of N. For 64×64 data size, modA-C and modB-C have the lowest storage requirements. In fact, if Architecture 2 is used to implement the encoder, the storage requirements are always lower. This is because in that implementation, the four sets of outputs in the last stage of the encoder are produced at the same time, thereby allowing the decoder to start processing immediately.

V. Conclusion

In this paper we have described SIMD array and folded implementations of the encoder and decoder based on the 2-D Discrete Wavelet Transform. The SIMD array implementations require N^2 processors (each processor consists of multiplier, adder and registers) and are suitable for multichip implementation or for mapping onto a parallel computer environment. The folded implementations, on the other hand, require only $4K$ multipliers and are suitable for single chip implementation. To support this statement, we present the results of an encoder design using Algorithm A for $N \times N = 64 \times 64$, $K = 4$, $T = 4$ and $J = 4$. The datapath consists of 16 8×8 array multipliers pipelined to 2 levels and 12 carry look ahead adders and has an area of 2.7 sq mm in .5 micron Motorola technology. The two memory modules have an area of 1.04 sq mm. The projected chip area is ≈ 5 sq mm. For large N, the chip area is clearly dominated by the area required by the memory modules. However, even for $N \times N = 512 \times 512$, the memory required is ≈ 4 KBytes, which is small enough to be implemented on the same chip as the datapath.

A comparison of the different implementations presented in this paper shows that Architecture 2 has lower forward-latency as well as lower storage requirements compared to Architecture 1. The reduction in forward-latency and storage is at the expense of 2 additional parallel computation units ($2K$ multipliers, where K is the filter size). This implies that Architecture 2 would only be preferred in applications where the area occupied by the $2K$ multipliers is less than the area required by N bytes of memory.

All the architectures presented in this paper assume that the borders have been padded with zeroes. If perfect reconstruction is important, then additional boundary data will have to be stored in each level of resolution. Our current work is geared towards developing 2-D DWT architectures that handle boundary data efficiently.

Acknowledgements

The authors would like to thank M. Vishwanath for many helpful comments during the initial phase of this work, and K. Johns-Vano for providing many of the hardware estimates.

References

[1] Aware, Inc., Cambridge, MA. *Aware Wavelet Transform Processor (WTP) Preliminary*, 1991.

[2] 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.

[3] C. Chakrabarti, M. Vishwanath, and R.M. Owens. A survey of architectures for the discrete and continuous wavelet transforms. *Journal of VLSI Signal Processing Systems*, pages 171-192, Nov 1996.

[4] I. Daubechies. The wavelet transform, time-frequency localization and signal analysis. *IEEE Trans. Info. Theory*, 36(5):961–1005, Sept 1990.

[5] T. Denk and K. Parhi. Calculation of minimum number of registers in 2-D discrete wavelet transforms using lapped block processing. *Int. Symp. on Circuits and Systems*, pages 77–81, 1994.

[6] F. Fridman and E. S. Manolakos. Distributed memory and control vlsi architectures for the 1-D discrete wavelet transform. *IEEE VLSI Signal Processing VII*, 1994.

[7] B.M. Gordon, T.H. Meng, and N. Chaddha. A 1.2mw video-rate 2D color subband decoder. *Int. Symp. on Solid State Circuits*, 1995.

[8] G. Knowles. VLSI architecture for the discrete wavelet transform. *Elec. Letters*, 26(15):1184–1185, Jul 1990.

[9] R. Lang, E.Plesner, H.Schröder, and A.Spray. An efficient systolic architecture for the one-dimensional wavelet transform. *Proceedings of SPIE conference on Wavelet Applications*, pages 925–935, Apr 1994.

[10] A.S. Lewis and G. Knowles. VLSI architecture for 2-D Daubechies wavelet transform without multipliers. *Elec. Letters*, 27(2):171–173, Jan 1991.

[11] S. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Trans. Pattern Analysis and Mach. Intell*, 11(7):674–693, July 1989.

[12] O. Rioul and P. Duhamel. Fast algorithms for wavelet transforms. *IEEE Trans. on Information Theory*, 38(2):569–586, Mar 1992.

[13] C. Chakrabarti and C. Mumford. Efficient Realizations of Analysis and Synthesis Filters based on the 2-D Discrete Wavelet Transform. *Int. Conf. on Acoustics, Speech and Signal Processing*, pages 3256-3259, 1996

[14] 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.

[15] A. Grzeszczak, M. K. Mandal. S. Panchanathan, and T. Yeap. VLSI Implementation of Discrete Wavelet Transform. *IEEE Trans. on VLSI Systems*, pp. 421-433, Dec 1996.

Chaitali Chakrabarti received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, in 1984, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland, College Park, in 1986 and 1990 respectively. Since August 1990 she has been in the Department of Electrical Engineering at Arizona State University where she is now an Associate Professor. Her research interests include VLSI architectures and algorithms for signal/image processing, low power system design, CAD tools for VLSI, and parallel algorithms and architectures.

Dr. Chakrabarti received the Research Initiation Award from the National Science Foundation in 1993 and a Best Teacher Award from the Arizona State University's College of Engineering and Applied Sciences in 1994. She is a member of the Center for Low Power Electronics (jointly funded by NSF, the state of Arizona and the industrial partners).

The photograph and biograph of C. Mumford is not available at this time.