EFFICIENT REALIZATIONS OF ANALYSIS AND SYNTHESIS FILTERS BASED ON THE 2-D DISCRETE WAVELET TRANSFORM

*Chaitali Chakrabarti** *Clint Mumford*

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

ABSTRACT

This paper presents folded architectures and scheduling algorithms for computing the 2-D DWT for analysis and synthesis filters. The folded architectures consist of two parallel computation units (one for computations along the rows and the other for computations along the columns) and two storage units to store the intermediate outputs that are generated by the two units. The scheduling algorithms range from those that try to minimize the latency to those that try to minimize the control unit complexity and keep the data flow regular. A comparison of the scheduling algorithms has been included to aid the designer in choosing an algorithm that is best suited for a particular application.

1. INTRODUCTION

The last few years have seen wavelet transforms being used for a wide range of applications: from signal analysis and signal coding to numerical analysis. This wide range of applications makes a study of their implementations in VLSI very important. In this paper we focus on the realizations of the separable 2-D Discrete Wavelet Transform (DWT) for analysis and synthesis filters. 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 exist only a handful of architectures for computing the 2-D DWT for the analysis filter [1],[2],[4],[5],[7]. The architecture of Lewis and Knowles [4] is tuned to the Daubechies 4-tap filter and as a result does not work efficiently for other wavelets. The architecture of Chakrabarti and Vishwanath [1] is for non-separable DWT and consists of 2 parallel computation units[1] of size K^2 and a storage unit of size $\approx NK$. The architecture of Vishwanath, Owens and Irwin [7] is for separable 2-D DWT and consists of 2 systolic arrays of size K, 2 parallel computation units of size K and a storage unit of size $\approx N(2K+J)$. In [7], 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 into 2 per 2 cycles output. The architecture of Denk and Parhi [2] 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 synthesis filter architecture that was designed by Gordon, Meng and Chaddha [3] for real-time video decompression. The architecture has been successfully implemented on a single chip and has a very low power consumption (1.2 mW at 1V supply).

In this paper we propose architectures and scheduling algorithms for the analysis and the synthesis filters that are based on the 2-D DWT. This work is an extension of our earlier work [6] on architectures and scheduling algorithms for 1-D DWT. We describe folded architectures where in each cycle only one data is input and one data is output. These architectures are well suited for single chip VLSI implementations. We propose scheduling algorithms that (i) try to minimize the latency and/or (ii) try to minimize the storage and keep the dataflow regular. We compare the performance of the scheduling algorithms with respect to latency, storage size and control unit complexity. This study gives the designer the flexibility to choose an algorithm that is suitable for a particular application.

The rest of the paper is organized as follows. In Section 2 we briefly describe the subband structure of the analysis and the synthesis filters. We describe the folded implementation and scheduling algorithms for the analysis filter and synthesis filter in Sections 3 and 4 respectively. We conclude the paper in Section 5.

2. PRELIMINARIES

We assume that the image size is $N \times N$, the number of stages is J, the high pass and the low pass filters are of size K and have a latency of T cycles. We refer to the latency of the analysis filter as the forward-latency and that of the synthesis filter as the reconstruction-latency. The analysis and the synthesis filters can be implemented by the filter bank structures shown in Figures 1 and 2.

In the 2-D DWT filter bank structure (Figure 1), 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 $(LL)^aL$ out-

*This work was supported in part by a grant from NSF, no. MIP-9309504.

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

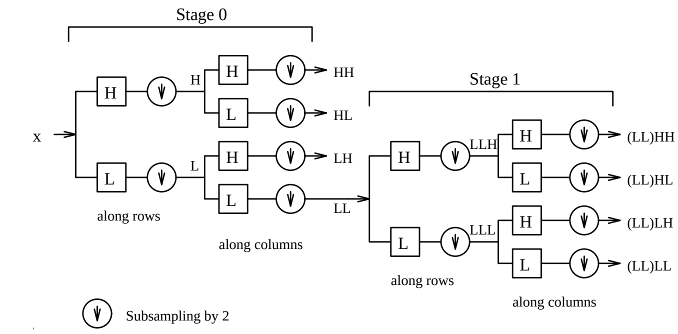

Figure 1. The 2-D DWT filter bank structure for $J = 2$ stages.

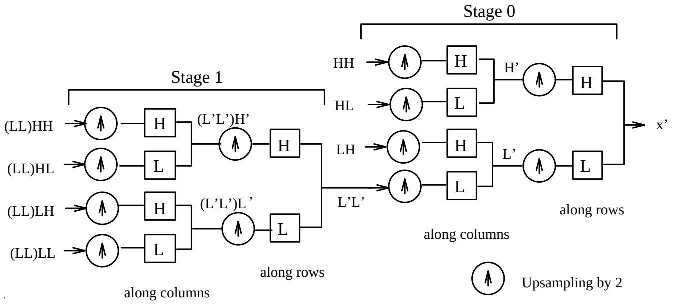

Figure 2. The 2-D IDWT filter bank structure for $J = 2$ stages.

put arrays is $\frac{N}{2^a} \times \frac{N}{2^{a+1}}$, $0 \leq a < J$. Similarly, $(LL)^a HH$ and $(LL)^a HL$ are the high pass and low pass outputs after computing along the columns of $(LL)^a H$, and $(LL)^a LH$ 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 $\frac{N}{2^a} \times \frac{N}{2^a}$, $0 \leq a < J$.

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

3. ANALYSIS FILTER IMPLEMENTATION

3.1. Folded Architecture

In this section we describe a folded architecture for the analysis filter which consists of two parallel computation units (Filters 1 and 2) which compute along the rows, two parallel computation units (Filters 3 and 4) which compute along the columns, and two storage units. Figure 3 gives the block diagram of this architecture.

Filter 1 computes the L and the H outputs along each row. Since the L outputs are computed every other cycle, the L and the H outputs can be computed in an inter-

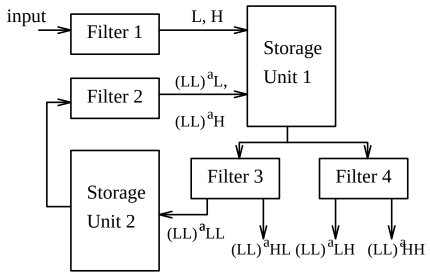

Figure 3. Block diagram of the architecture for 2-D DWT

leaved fashion. **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 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. **Filter 4** computes the outputs $(LL)^a LH$, and $(LL)^a HH$ along the columns, $0 \leq a < J$. It first computes the outputs of $(LL)^a HL$ and then (N cycles later) computes the outputs of $(LL)^a HH$.

3.2. Scheduling Algorithms

The scheduling procedure that is followed by the two algorithms (A and B) are described below. The algorithms 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 $(LL)^{a-1} HL_{i,j}$ N cycles after $(LL)^a_{i,j}$, $0 \leq i, j < \frac{N}{2^a}$.
   Schedule $(LL)^{a-1} HH_{i,j}$ at the same time as $(LL)^{a-1} HL_{i,j}$.
4. for $a = 1$ to $J - 1$ schedule $(LL)^a L_{i,j}$ T cycles after $(LL)^a_{i,2j}$, $0 \leq i < \frac{N}{2^a}$, $0 \leq j < \frac{N}{2^{a+1}}$.
   Schedule $(LL)^a H_{i,j}$ N cycles after $(LL)^a L_{i,j}$.

Algorithm A: The primary goal of this scheduling algorithm is to keep the dataflow regular 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 $(LL)^a_{i,j-1}$. The key step of this procedure is determining a conflict-free schedule for $(LL)^a_{0,0}$, $0 \leq a < J$, since this guarantees that there will be no conflicts for all other outputs of that stage.
Algorithm B: The primary goal of this algorithm is to reduce the forward-latency at the expense of a mild increase

Forward-latency:

Alg.	T=3	T=4	T=5
A	31	39	43
B	24	32	40
min. poss.	24	32	40

Complexity of the control circuit:

Alg.	T=3	T=4	T=5
A	very low	very low	very low
B	moderate	moderate	low

Figure 4. Comparison of the forward-latency and the complexity of the control unit in the analysis filter for different values of T when $J = 4$ and the data size is 64×64

Alg.	NxN=64x64			
	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)

Figure 5. Comparison of the storage required in the analysis filter for different values of K and T when $J = 4$ and $N = 64$. Notation Z(X+Y): Z is the total storage, X is the size of Storage Unit 1, and Y is the size of Storage Unit 2.

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. 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.

3.3. Comparison

Figure 4 compares the performance of the two algorithms with respect to forward-latency and complexity of the control unit. The forward-latency is defined as the difference in the time between when the first x-value is input and when the first $(LL)^J$ value is output. 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 5 compares the storage that is required in the analysis filter for different values of K and T. The storage calculations are based on lifetime analysis. Note that the storage requirements of the implementations based on Algorithms A and B are similar.

4. SYNTHESIS FILTER IMPLEMENTATION

4.1. Folded Architecture

In this section we describe a folded architecture for the synthesis filter which consists of two parallel computation units (Filters 1 and 2) which compute along the rows, two parallel computation units (Filters 3 and 4) which compute along the columns, and two storage units. The block diagram of this architecture is given in Figure 6. The input is read in from the analysis filter memory in row major order by Filters 1 and 2, and the output x' is generated in row major

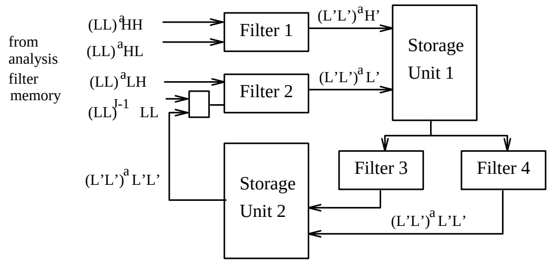

Figure 6. Block diagram of the architecture for 2-D IDWT

order by Filters 3 and 4.

Filter 1 is a column filter which reads data $(LL)^aHH$ and $(LL)^aHL$ from the analysis filter memory and computes the outputs $(L'L')^aH'$, $J > a \geq 0$. 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 $(L'L')^aH'$, Filter 1 reads in the same data twice. **Filter 2** is also a column filter which reads data $(LL)^aLH$ and $(LL)^{J-1}LL$ from the analysis filter memory and $(L'L')^aL'L'$ from Storage Unit 2, $J-1 > a \geq 0$. It reads the input data along the columns and computes the outputs $(L'L')^aL'$ in row major order. The multipliers in Filters 1 and 2 are programmable, since the same input data is used to produce the two outputs at two different times. **Filters 3 and 4** are row filters that read $(L'L')^aH'$ and $(L'L')^aL'$ 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 $(L'L')^a_{i,2j+1}$. The two outputs are then written into Storage Unit 2. Filters 3 and 4 consist of a fixed set of K multipliers. While the multipliers in Filter 3 consist of the odd-indexed coefficients of the high pass and the low pass filters, the multipliers in Filter 4 consist of the even-indexed coefficients of the high pass and the low pass filters.

4.2. Scheduling Algorithms

In this section we describe two algorithms for scheduling the operations in the architecture. Both algorithms generate the x' outputs in a regular manner. Specifically, 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).

Algorithm C: The primary goal of this scheduling algorithm is to keep the dataflow regular and reduce the complexity of the control unit. This comes at the expense of an increase in the reconstruction-latency. The outputs are scheduled in a regular manner. For instance, $(L'L')^aL'_{i,j}$ is scheduled 2^{a+1} cycles after $(L'L')^aL'_{i,j-1}$. 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 $(L'L')^aL'$ outputs, $1 \leq a < J$. Once this

Reconstruction-latency:

Alg.	T=3	T=4	T=5
C	31	39	43
D	21	28	35
min. poss.	21	28	35

Complexity of the control circuit:

Alg.	T=3	T=4	T=5
C	very low	very low	very low
D	moderate	low	low

Figure 7. Comparison of the reconstruction-latency and the complexity of the control unit in the synthesis filter for different values of T when $J = 4$ and the data size is 64×64.

scheduling is determined, the schedule of the other outputs are determined in a straightforward manner.

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 size of the storage unit and the complexity of the control unit. In this procedure low reconstruction-latency is achieved by scheduling the $(L'L')^a_{0,0}$ outputs first, $0 \le a < J$. 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.

For systems where the synthesis filter starts computing before the analysis filter has finished all the computations, the IDWT schedule is delayed if there are any conflicts with the DWT schedule.

4.3. Comparisons

Figure 7 compares the performance of the two algorithms with respect to reconstruction-latency and complexity of the control unit. 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 analysis filter outputs are available at the time the synthesis filter starts its computation. Note that Algorithm D requires significantly smaller reconstruction-latency compared to Algorithm C, and would clearly be the choice in low latency applications. The control circuitry to implement the schedule of Algorithm D is more complex than that required to implement the schedule of Algorithm C. This is because the schedule of Algorithm D is not as regular as that of Algorithm C. Figure 8 compares the storage that is required in the synthesis filter for different values of K and T. Note that the storage requirements are almost identical for the implementations based on Algorithms C and D.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have described folded architecture implementations of analysis and synthesis fiters that are based on 2-D DWT. We have proposed two scheduling algorithms for each of the folded implementations. The first scheduling algorithm tries to make the dataflow regular (a feature that results in easier VLSI implementation) while the second algorithm tries to reduce the latency (a feature that is desirable in video decoders). The synthesis filter implementation in this paper assumes that all the outputs of the analysis filter are available before the decoding begins. We are currently designing an analysis-synthesis system where the system-latency (difference in time between when the first reconstructed data x' is output and when the first input data x is input) is minimum. Minimization of system-latency is very important for interactive video applications. In order to reduce the system-latency, the folded architecture of the analysis filter that has been described in Section 3 has to be modified. The modification should be such that the $(LL)^a LL$ and the $(LL)^a LH$ outputs are computed at the same time (and not N cycles apart). This would ensure reduction in system-latency by N cycles, since the synthesis filter can start decoding N cycles earlier.

Alg.	NxN=64x64			
	K=4,T=3	K=4,T=4	K=6,T=3	K=6,T=4
C	124(12+112)	126(14+112)	186(18+168)	188(20+168)
D	126(14+112)	126(14+112)	186(18+168)	188(20+168)

Figure 8. Comparison of the storage required in the synthesis filter for different values of K and T when $J = 4$ and $N = 64$. Notation Z(X+Y): Z is the total storage, X is the size of Storage Unit 1, and Y is the size of Storage Unit 2.

REFERENCES

[1] 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.

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

[3] 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.

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

[5] T.H. Meng, B.M.Gordon, E.K. Tsern, and A.C. Hung. Portable video-on-demand in wireless communication. *Proceedings of the IEEE*, 83:659–680, April 1995.

[6] M. Vishwanath and C. Chakrabarti. A VLSI architecture for real-time hierarchical encoding/decoding of video using the wavelet transform. *Int. Conf. on Acoustics, Speech and Signal Processing*, pages 401-404, 1994.

[7] 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.

EFFICIENT REALIZATIONS OF ANALYSIS AND SYNTHESIS FILTERS BASED ON THE 2-D DISCRETE WAVELET TRANSFORM

Chaitali Chakrabarti[2] *and Clint Mumford*

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

This paper presents folded architectures and scheduling algorithms for computing the 2-D DWT for analysis and synthesis filters. The folded architectures consist of two parallel computation units (one for computations along the rows and the other for computations along the columns) and two storage units to store the intermediate outputs that are generated by the two units. The scheduling algorithms range from those that try to minimize the latency to those that try to minimize the control unit complexity and keep the data flow regular. A comparison of the scheduling algorithms has been included to aid the designer in choosing an algorithm that is best suited for a particular application.