ISCAS 2000 - IEEE International Symposium on Circuits and Systems, May 28-31, 2000, Geneva, Switzerland

A Programmable VLSI Architecture for 2-D Discrete Wavelet Transform

Chien-Yu Chen, Zhong-Lan Yang, Tu-Chih Wang, Liang-Gee Chen

Department of Electrical Engineering, National Taiwan University
1, Roosevelt Road Sec. 4, Taipei 106, Taiwan

Abstract

Many VLSI architectures for computing the discrete wavelet transform (DWT) were presented, but the parallel input data sequence and the programmability of the 2-D DWT were rarely mentioned. In this paper, we present a parallel-processing VLSI architecture to compute the programmable 2-D DWT, including various wavelet filter lengths and various wavelet transform levels. The proposed architecture is very regular and easy for extension. To eliminate high frequency components, the pixel values outside the boundary of the image are mirror-extended as the symmetric wavelet transform and the mirror-extension is realized via the routing network. This design has been implemented and fabricated in a 0.35μm 1P4M CMOS technology and the working frequency is 50 MHz. The chip size is about 5200μm×2500μm. For a 256×256 image, the chip can perform 30 frames per second with the filter length varying from 2 to 20 and with various levels. Hence the proposed architecture is suitable for real-time applications such as JPEG 2000.

1. Introduction

The DWT is a multi-scale frequency analyzer. It decomposes a signal into many subbands with different frequency characteristics. The higher frequency subbands have finer time resolution, and the lower frequency subbands have coarser time resolution. Since it outperforms some traditional time-frequency representations such as the short-time Fourier transform, the DWT has been considered suitable in many image coding applications. But in real-time applications the DWT must operate under high speed, therefore many VLSI architectures have been reported to achieve real-time computing and high hardware utilization. Knowles [1] presented a simple 2-D DWT architecture for the case of Daubechies 4-tap wavelet. Later, Parhi [3] proposed two architectures referred to the folded architecture and the digit-serial architecture for 1-D and 2-D DWT. Vishwanath [4] improved the performance by means of the linear systolic array, and presented an efficient storage structure.

However, few of these papers discussed about the programmability of the DWT such as reconfiguring the filter length. Actually, previous studies are difficult to adaptively modify their data paths to meet various wavelet filter lengths. The proposed architecture uses the routing network torealize the DWT with various filter lengths and various transform levels. Also, most of the previous studies only considered the serial input sequence, however it is very likely to have the parallel input sequence in an embedded system, such as a digital still camera system embedded with a wavelet-based compression engine. Our design methodology, which chooses the parallel processing, is based on the above point of view.

2. Row-Based RPA

The RPA, which is widely used in the DWT, can not be directly applied to our parallel-processing architecture. Since in our architecture an entire row of the image data must be processed in parallel, we need to schedule an entire row, rather than a single pixel, each time. Therefore the scheduling can be viewed as the 1-D RPA [7], in which each pixel can be viewed as an entire row of the wavelet coefficients. From the down-sampling characteristic of the DWT, we can see that the number of rows in the higher octave is one half the number of rows in the lower octave. Thus we can intersperse the row operation of various octaves in between the first octave like the 1-D RPA. In addition, after each row operation, we can proceed the column operation to generate the wavelet coefficients of an entire row in the corresponding level. If we schedule the row and column operations like the above described, the scheduling will be very simple and straightforward. Table 1 shows the row-based RPA.

Time	Operation
0	Octave1 Row + Octave1 Column
1	Octave2 Row + Octave2 Column
2	Octave1 Row + Octave1 Column
3	Octave3 Row + Octave3 Column
4	Octave1 Row + Octave1 Column
5	Octave2 Row + Octave2 Column
6	Octave1 Row + Octave1 Column
7	-
⋮	⋮

Table 1. The row-based RPA.

It should be noted that only $1/2^{(i-1)}$ of the PEs operate in the higher octaves, i is the octave index. Although the operations in higher octaves have lower hardware utilization, they also take less processing time ($1/2^i$ of overall processing time). We can estimate the overall hardware utilization:

$$
\begin{aligned}
Utilization &= \sum_{i=1}^{L}(\text{level}i\text{\_utilization} \times \frac{\text{level}i\text{\_operation\_time}}{\text{overall\_operation\_time}}) \\
&= 1\times\frac{1}{2}+\frac{1}{2}\times\frac{1}{2^2}+\frac{1}{2^2}\times\frac{1}{2^3}+\ldots+\frac{1}{2^{L-1}}\times\frac{1}{2^L} \\
&= \frac{2}{3}(1-\frac{1}{4^L}) \\
&= \frac{2}{3} \qquad \text{if } L \text{ is large}
\end{aligned}
$$

Compared with the conventional pyramid algorithm, the row-based RPA can reach the same hardware utilization, 66.7%.

3. Architecture

Considering a 3-level DWT and the filter length is 3, the image data and the wavelet coefficients are shown in Figure 1. The lowest blocks are the image data, while the higher blocks are wavelet coefficients in different octaves and the dash lines illustrate the boundary of the image. The transparent blocks are those required for the operations in the next octave. For an image with 8 pixels (*a* ~*h*) per row (see Figure 1), we need 8 processing elements (PEs) for parallel processing, and each PE computes one wavelet coefficient in the corresponding column. Those blocks outside the boundary (*b* and *g*) are mirror-extended with *a* and *h* respectively. Operations in the higher octaves are similar to those in the first octave.

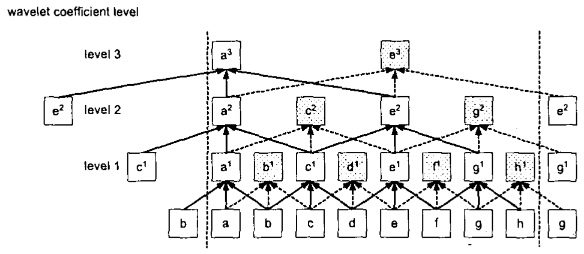

Figure 1. Operations of an 8-point wavelet transform.

The DWT operations are implemented with the proposed architecture shown in Figure 2. The architecture has five main parts: the PEs, the routing network, the memory cells (40 rows×256×12 bits), the addressing elements and a controller. A PE is shared by all memory cells (one memory cell stores one bit data) in the corresponding column, i.e. one PE per column. After calculation, the PE stores the wavelet coefficients in the memory cells under them. The detailed function of each part will be given in the following sections.

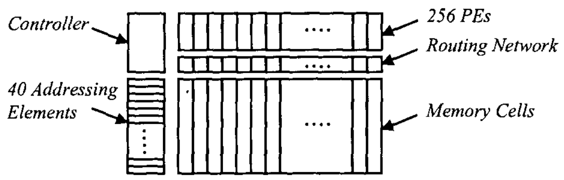

Figure 2. Block diagram of the proposed DWT architecture.

3.1 DWT Operations

The basic idea of this architecture is to compute the DWT by the SIMD parallel-processing concept with the efficient routing network. The separable 2-D DWT can be divided into two operations, the row and column operations. The whole 2-D wavelet operation is shown in Figure 3. Initially, the input data buffer receives the serial data and prepares for parallel processing. After receiving a whole row, i.e. 256 pixels, it stores them to the memory and the row operation starts. First, the routing network reads a whole row and then dispatches the data to the PEs to calculate the wavelet coefficients. The column operation, which is almost the same for the PEs, follows the row operation. The only difference is that during the column operation, the PEs get the data in the same column because the data generated by the row operation are stored in the same column but in the different row. After the row and column operations, only the wavelet coefficients in the LL band are stored for the operations in the higher octaves and those in the other three subbands (LH, HL and HH) are output to reduce the storage size.

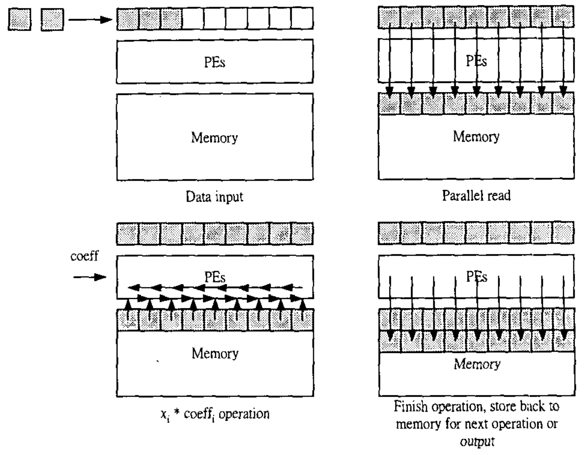
PEs
Memory
Data input
PEs
Memory
Parallel read
coeff
PEs
Memory
x_i * $coeff_i$ operation
PEs
Memory
Finish operation, store back to memory for next operation or output

Figure 3. DWT operations.

3.2 Bit-Serial Processing Elements

The PEs are the main part of this design for the parallel-processing architecture. They get the image data from the routing network to compute the wavelet coefficients. From the hardware C simulation, the peak signal-to-noise ratio (PSNR) saturates when the word lengths of the intermediate data and the wavelet filter coefficients are 12 bits wide. Although each PE is responsible for one wavelet coefficient in a row, a PE's pitch can only match the total pitch of four memory cells. For the regularity in the full-custom layout, we divide 12 bits into 3 rows and use a 4b (filter coefficient)×1b (data) MAC instead of a full-precision, i.e. 12b×12b, MAC to compute the wavelet coefficient by time multiplexing. As shown in Figure 4, each PE consists of four 1-bit full adders, a 24-bit DRAM buffer, and some logic circuits for the multiplication.

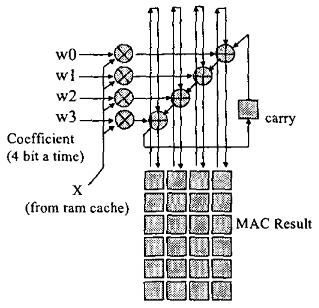

Figure 4. The processing element.

The computing process for a wavelet coefficient is as follows.

$$y_i^j = \sum_{k=-(L-1)/2}^{(L-1)/2} y_{i-k}^{j-1} \times w_k = \sum_{m=0}^{11}\Big(\sum_{k=-(L-1)/2}^{(L-1)/2} y_{i-k}^{j-1}[m] \times w_k \Big)$$
$$= \sum_{k=-(L-1)/2}^{(L-1)/2} y_{i-k}^{j-1}[0] \times w_k + \sum_{k=-(L-1)/2}^{(L-1)/2} y_{i-k}^{j-1}[1] \times w_k + .. + \sum_{k=-(L-1)/2}^{(L-1)/2} y_{i-k}^{j-1}[11] \times w_k$$

where

$$\sum_{k=-(L-1)/2}^{(L-1)/2} y_{i-k}^{j-1}[0] \times w_k$$
$$= \underline{y_i^{j-1}[0] \times w_0} + y_{i-1}^{j-1}[0] \times w_1 + y_{i+1}^{j-1}[0] \times w_{-1} + \ldots + y_{i-(L-1)/2}^{j-1}[0] \times w_{(L-1)/2}$$
$$= \underline{y_i^{j-1}[0] \times w_0[3:0] + y_i^{j-1}[0] \times w_0[7:4] + y_i^{j-1}[0] \times w_0[11:8]} +$$
$$\ldots + y_{i-(L-1)/2}^{j-1}[0] \times w_{(L-1)/2}[11:8]$$

First, complete all LSBs of the data; then proceed to the second LSBs and continue to finish a full-precision MAC operation. It needs 72 cycles to compute an MAC operation and 72*n* cycles to complete a wavelet coefficient. The clock frequency for a 256x256 image at 30 frames per second is

$$\sum_{i=0}^{\infty}\left(72n \times \frac{256}{2^i} \times 2\right) \times 30 = 2.2n \text{ MHz}$$

where *n* is the maximum filter length of HPF and LPF.

3.3 Routing Network

The routing network is in charge of dispatching the correct data at the right moment to each PE. In the row operation, the routing network reads a whole row of data (256×4 bits) from the memory and provides the PEs with the data by two shift registers. Figure 5 shows the architecture of the routing network. Each routing network unit consists of three registers and some multiplexers and there is one routing network unit per column. The connection between the routing network units forms two shift registers (one shifting right, while the other shifting left). In the beginning of a row operation, all routing network units read one bit data from the memory and transfer it to the corresponding PEs to start the computing process. Every time the routing network shifts, the PEs can get the neighboring data. The longer the wavelet filter length is, the more times the routing network will shift to get the data required for the DWT. For operations in the higher octaves, the PEs will require (2^{i-1})th data away from both sides of the current column and the routing network will shift 2^{i-1} times before the PEs finish the current operation. Although it takes more time to finish a wavelet operation with longer filter length in this architecture, the programmability can be easily achieved.

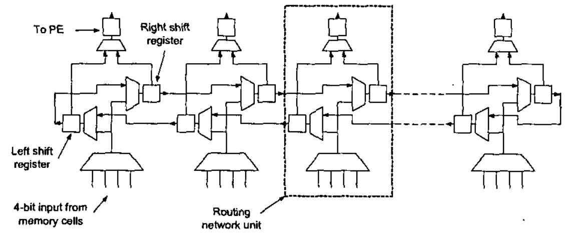

Figure 5. The routing network.

Another important issue for the DWT is the boundary problem. To eliminate high frequency components in the boundary of an image, it is necessary to mirror-extend the image data before filtering. The connection in the boundary of the routing network is also shown in Figure 5. Since these two shift registers connect like a ring, if the value in the boundary shifts left, it will shift back to the right-shifting registers, and vice versa. Through the connection, it seems that the image data are mirror-extended in the boundary even we don't really extend the image data. We could implement the mirror-extension as long as the filter length is shorter than twice the length of a row.

3.4 Addressing Elements and Memory Cells

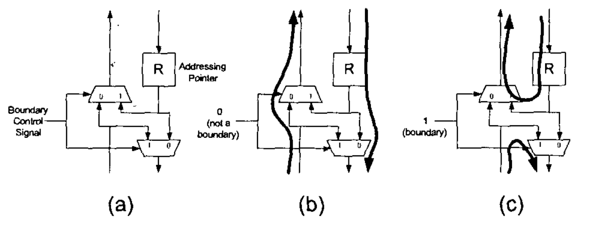

Figure 6. Simplified diagram of the addressing element.

For the memory addressing, each row of the memory has an addressing element, which contains a register and some logic circuits as shown in Figure 6(a). The reason we use addressing elements instead of a row decoder is for the programmability and expandability. Owing to the regular data access in the DWT, some shifting operations can afford this job. By programming the addressing elements, we can divide the memory into many memory banks for different octaves. If the filter length is K and the number of octaves is L, the memory cells will be divided into L banks and each bank contains (K+1) rows of the memory, in which K rows are for the column operation, and one row is for the row operation. If the addressing element is not in the boundary of a memory bank, it will shift down the addressing pointer to the next register as shown in Figure 6(b). Otherwise the addressing pointer will be shifted back to the addressing element in the opposite boundary to form a circular shift register, as shown in Figure 6(c). If an addressing element is activated, the corresponding row of the memory cells will be read or written. While expanding the number of rows of the memory, we just need to add an addressing element to the left of each row. For the considerations of area and implementation, each memory cell is a 3-transistor (3T) DRAM. The area of a 3T DRAM is less than a half of a 6T SRAM, and it is simpler to design than a 1T DRAM.

4. Comparison

Table 2 lists some of the existing DWT architectures and N is the image size. The proposed architecture doesn't have any full-precision MAC and requires only 4N 1-bit full adders to achieve 12b×12b precision. Besides, other architectures require at least K MACs to implement the DWT and their data paths depend on the filter length, so it is difficult for them to compute the DWT with various filter lengths. The characteristic of the programmability in this design is at the cost of the memory utilization and the storage size, but it can compute the DWT with various filter lengths and various transform levels. The boundary problem for the DWT is also important, but most literatures didn't mentioned it before. Using the proposed architecture, the boundary problem is solved by the mirror-extension, which is independent of the filter length, via the efficient routing network. Since the proposed architecture is very regular, it is easy to extend the PE array and the memory cells for a larger image and more octaves.

5. Conclusion

In this paper, we have proposed an SIMD-based VLSI architecture to compute the programmable 2-D DWT. This chip contains 256 PEs in order to calculate 256 pixels simultaneously. With 40 rows of memory cells, this chip can compute 4-tap 2-D DWT with 8 levels, or 9-tap 2-D DWT with 4 levels, even up to 16-tap 2-D DWT with 2 levels. In the coming standard, JPEG2000, there are various filters suggested and the proposed architecture can compute the DWT with all suggested filters for different applications. The chip layout is show in Figure 7 and the chip specification is listed in Table 3.

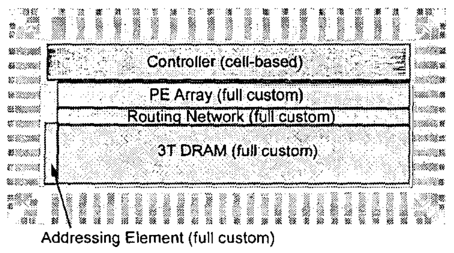

Figure 7. The whole chip layout.

References

[1] G. Knowles, "VLSI Architecture for the Discrete Wavelet Transform," *Electronics Letters*, vol. 26, pp. 1184-1185, July 1990.

[2] M. Vishwanath, R. M. Owens and M. J. Irwin, "VLSI Architectures for the Discrete Wavelet Transform," *IEEE Trans. on Circuits and Systems-II*, vol. 42, no. 5, pp. 305-316, May 1995.

[3] K. K. Parhi and T. Nishitani, "VLSI Architecture for Discrete Wavelet Transforms," *IEEE Trans. on VLSI Systems*, vol. 1, no. 2, pp. 191-202, June 1993.

[4] R. M. Owens and M. Vishwanath, "A Very Efficient Storage Structure for DWT and IDWT Filters," *Journal of VLSI Signal Processing*, vol. 19, pp. 215-225, 1998.

[5] C. Chakrabarti, and M. Vishwanath, "Efficient Realizations of the Discrete and Continuous Wavelet Transforms: From Single Chip Implementation to Mappings on SIMD Array Computers," *IEEE Transactions on Signal Processing*, vol. 43, no. 3, pp. 759-771, March 1995.

[6] C. Yu and S. J. Chen, "Efficient VLSI Architecture for Separable 2-D Discrete Wavelet Transforms," *IEEE International Symposium on Consumer Electronics*, Oct 1998.

[7] M. Vishwanath, "The Recursive Pyramid Algorithm for the Discrete Wavelet Transform," *IEEE Trans. on Signal Processing*, vol. 42, no. 3, pp.673-676, Mar. 1994.

Architecture	MAC	FA	Routing	Memory	Algorithm	Programmable	Boundary	Expendable
Ours	-	4N	Simple	(K+1)NL	Non-separable	Yes	Mirror-extended	Easy
Par-Par [6]	6K	-	Simple	2NK	Separable	No	No	Moderate
Sys-Par [2]	4K	-	Simple	2NK	Separable	No	No	Moderate
2D filter [5]	K^2	-	Complex	2NK	Non-separable	No	No	Complex
Direct approach [2]	K	-	Simple	N^2	Separable	No	No	Complex

Table 2. Comparison of various architectures.

Parameter	Value
Frame size	256×256
Frame rate	30 frames/sec
Wavelet tap number	2 to 20
Technology	TSMC 0.35μm CMOS 1P4M
Number of transistors	545403 (with 122880 3T cells)
Number of IO pads	65
Die size	4350μm × 1800μm
Core size	5200μm × 2500μm
Package	68 PGA
Clock rate	50MHz
Voltage	3.3V
Power	520mW

Table 3. Chip specification.