Dirasat, Pure Sciences, Volume **32**, No. **1**, 2005

A Parallel FPGA Implementation of the Discrete Wavelet Transform Based on Polyphase Decomposition and Distributed Arithmetic Techniques

***Ali Al-Haj* ***

ABSTRACT

ASIC hardware implementations of the discrete wavelet transform are required to cope with the intensive real-time computations of the wavelet transform. In this paper, we describe a parallel implementation of the wavelet transform using one type of programmable Application Specific Integrated Circuits (ASICs); Filed Programmable Gate Arrays (FPGAs). The implementation is based on reformulating the discrete wavelet transform using the distributed arithmetic and ploy phase decomposition techniques, so that the ample inherent parallelism of the transform can be well exploited by the fine-grained parallel architecture of Virtex FPGAs. The implementation is simple, scalable, and performs both the forward and inverse discrete wavelet transforms. Performance results demonstrate the applicability of FPGAs with the distributed arithmetic and ploy phase decomposition techniques to achieve the required high computational speeds of the discrete wavelet transform.

KEYWORDS: Discrete Wavelet Transform, Mallat's Pyramid Algorithm, FPGA Virtex Devices, Polyphase Decomposition, Distributed Arithmetic, Parallel Implementation.

INTRODUCTION

The discrete wavelet transform is a powerful new mathematical method with a broad spectrum of potential applications (Burrus et al., 1998). It has already been used successfully in signal processing (Rioland Vetterli, 1991) and numerical analysis (Beylkin et al., 1992), among many other audio-visual applications. In particular, the area of data compression has benefited incredibly from the wavelet transform (Ebrahimi and Pereira, 2002). However, the transforms high computational requirements may limit its wide-spread, especially in applications requiring real time performance. Consequently, there has been a great demand on high-speed computing devices to meet the real-time computational requirements of the transform.

Fortunately, the discrete wavelet transform is inherently parallel, and it lends itself to hardware implementations on VLSI custom devices (Smith, 1997). Indeed, many custom VLSI-based Application Specific Integrated Circuits (ASICs) implementations of the discrete wavelet transform have appeared in literature (Chakabarti et al., 1996; Knowles, 1990; Parhi and Nishitani, 1993 and Chakabarti and Vishwanath, 1995), however, most of the proposed architectures require complex control units, and are not easily scaled up for different wavelets filters and different octave levels. Recently, Filed Programmable Gate Arrays (FPGAs) have become an attractive implementation platform for many digital signal processing systems (Seals and Whapshott, 1997; Kollig et al., 1997; Shand, 1995 and Villasenor et al., 1995). They are essentially programmable ASICs which offer intermediate capabilities between those offered by custom VLSI ASICs and programmable digital signal processors. Indeed, programmability of FPGAs makes them a perfect choice for implementing the discrete wavelet transform since this would allow easy modification of different wavelet types and design parameters, such as number of

* Princess Sumaya University, Amman, Jordan. Received on 21/11/2003 and Accepted for Publication on 5/7/2004.

wavelet levels, filters specifications, and bit precision of wavelet coefficients.

In this paper, we describe a parallel FPGA implementation of the discrete wavelet transform using the highly replicated, register rich, Virtex FPGAs (Xilinx Corporation, 1998). The implementation is based on achieving high execution speeds by exploiting the abundant inherent parallelism of the wavelet transform using the distributed arithmetic (White, 1989) and plolyphse decomposition (Bellanger et al., 1976) techniques. Unlike most reported implementations which concentrate on architectural development, this implementation describes the actual implementation of both the forward and inverse transforms. The paper is organized as follows. Section 2 gives an overview of the Mallat's pyramid algorithm which is used to compute the coefficients of the discrete wavelet transform. The implementation is described in detail in section 3, and simulated in section 4. Performance results are presented and compared in section 5. Finally, section 6 presents some concluding remarks.

2. Mallat's Pyramid Algorithm

Wavelets are special functions which, in a form analogous to sines and cosines in Fourier analysis, are used as basal functions for representing signals . The coefficients of the Discrete Wavelet Transform (DWT) can be calculated recursively and in a straight forward manner using the well-known Mallat's pyramid algorithm (Mallat, 1989). Based on Mallat's algorithm, the discrete wavelet coefficients of any stage can be computed from the coefficients of the previous stage using the following iterative equations:

$$W_L(n,j)=\sum_m W_L(m,j-1)\,h_0(m-2n) \quad (1)$$

$$W_H(n,j)=\sum_m W_L(m,j-1)\,h_1(m-2n) \quad (2)$$

Where $W_L(n,j)$ is the n^{th} scaling coefficient at the j^{th} stage, $W_H(n,j)$ is the n^{th} wavelet coefficient at the j^{th} stage, and $h_0(n)$ and $h_1(n)$ are the dilation coefficients corresponding to the scaling and wavelet functions, respectively. In order to reconstruct the original data, the DWT coefficients are upsampled and passed through another set of low pass and high pass filters, which is expressed as

$$W_L(n,j)=\sum_k W_L(k,j+1)\,g_0(n-2k)+\sum_l W_H(l,j+1)\,g_1(n-2l) \quad (3)$$

where $g_0(n)$ and $g_1(n)$ are respectively the low-pass and high-pass synthesis filters corresponding to the mother wavelet. It is observed from Equation (3) that the j^{th} level coefficients can be obtained from the $(j+1)^{th}$ level coefficients.

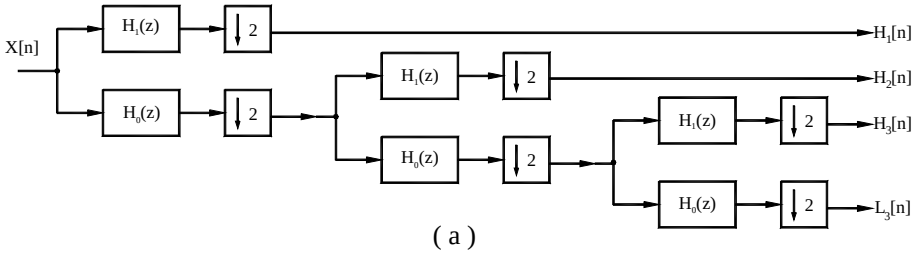
Daubechies 8-tap wavelet has been chosen for this implementation. This wavelet type is known for its excellent special and spectral localities which are useful properties in image compression (Daubechies, 1988). The filters coefficients corresponding to this wavelet type are shown in Table (1). H_0 and H_1 are the input decomposition filters and G_0 and G_1 are the output reconstruction filters.

3. The Implementation

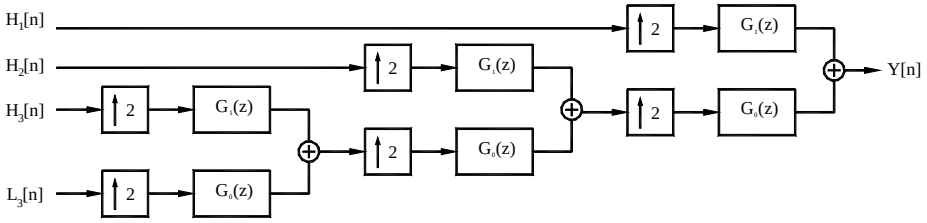
In this section, we describe an efficient implementation of the discrete wavelet transform. The implementation is based on rearranging the Quadratic Mirror Filter (QMF) discrete wavelet transform multirate structure, described in Strang et al. (1996) and shown in Figure 1, as a polyphase tree, and then implementing each sub-filter of the polyphase tree as a distributed arithmetic filter. This combination of the two efficient filter structures; polyphase and distributed arithmetic based structures, leads to an efficient implementation of the discrete wavelet transform. In what follows, we first describe the polyphase reformulation of the analysis and synthesis filter banks, and then we describe the distributed arithmetic implementation of the sub-filter structure of the polyphase filter banks.

Table 1. Daubechies 8-tap wavelet filter coefficients.

Filter	0	1	2	3	4	5	6	7
H_0	-0.0106	-0.0329	0.0308	0.1870	-0.0280	-0.6309	0.7148	-0.2304
H_1	0.2304	0.7148	0.6309	-0.0280	-0.1870	0.0308	0.0329	-0.0106
G_0	-0.2304	0.7148	-0.6309	-0.0280	0.1870	0.0308	-0.0329	-0.0106
G_1	-0.0106	0.0329	0.0308	-0.187	-0.0280	0.6309	0.7148	0.2304



(a)



(b)

Fig. 1. Mallat's Quadratic Mirror Filter Trees: (a). Forward DWT Tree; (b). Inverse DWT Tree.

3.1 Polyphase Filter Banks

Filters used in the parallel pyramid tree architecture of Figure 1, are constructed using FIR filters because of their inherent stability (Oppenheim and Schafer, 1999). A direct form realization of FIR filter is shown in Figure 2a. A computationally efficient realization of the filter consists of decomposing the filter into two sub-filters executing in parallel. This realization is based on the polyphase decomposition algorithm (Vaidyanathan, 1993), and results in a parallel architecture useful for real time applications. Consider the transfer function given in Equation (4). By separating the even numbered coefficients from the odd numbered ones, we can derive the polyphase form of the transfer function as follows:

$$H(z)=\sum_{k=0}^{N-1} h[k]\,z^{-k} \qquad (4)$$

$$H(z)=\sum_{n=0}^{M-1} h(2n)\,z^{-2n}+\sum_{n=0}^{M-1} h(2n+1)\,Z^{-(2n+1)} \qquad (5)$$

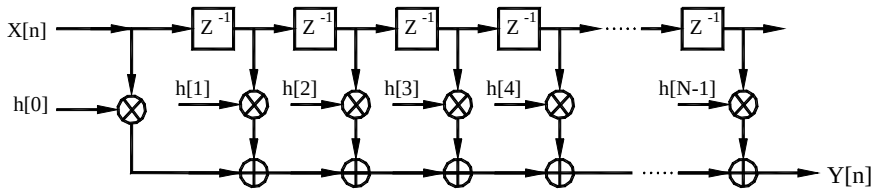
Where N represents the length of the filter, and M is equal to $N/2$ and represents number of even or odd filter coefficients. Now, let's define the abbreviations:

$$E_0(z)=\sum_{n=0}^{M-1} h(2n)\,z^{-n},\quad E_1(z)=\sum_{n=0}^{M-1} h(2n+1)\,z^{-n} \qquad (6)$$

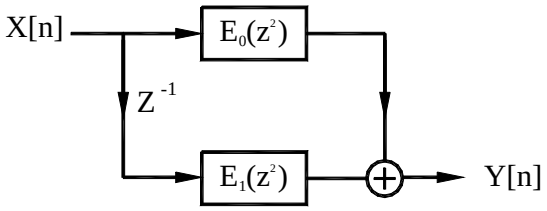
we can rewrite $H(z)$ in the form:

$$H(z) = E_0(z^2)+z^{-1}E_1(z^2) \qquad (7)$$

where $E_0(Z^2)$ and $E_1(Z^2)$ are the even sub-filter and odd sub-filter, respectively. Figure 2b shows the two-branch parallel realization of the direct-form FIR shown in Figure 2a.



(a)



(b)

Fig. 2. Polyphase Realization of the FIR Filter : (a).Original Direct Structure; (b). Polyphase Structure.

3.1.1 Analysis Filter Bank

The analysis filter bank shown in Figure 3a represents the basic building block of the forward discrete wavelet transform. It consists of two decimators connected in parallel; the upper decimator is a low pass filter followed by a down sampler, and the lower decimator is a high pass filter followed by a down sampler. The down-sampler operates by taking a filtered sequence *x[n]* and generating an output sequence *y[n]* according to the relation *y[n] = x[2n]*.

All filtered elements in the subsequence *x[2n+1]* are discarded. Consequently, the direct structure shown in Figure 3a is computationally inefficient since it unnecessarily calculates the values *x[n]* for $n \neq 2n$, which are discarded by the down sampler after being calculated. To avoid such unnecessary calculation, a more efficient, but equivalent, implementation of the analysis filter bank exists, and it is based on the concept of polyphase decomposition. If we represent the low pass transfer function in polyphase form, as explained earlier, we get:

$$H_0(z) = E_0(z^2) + z^{-1} E_1(z^2), \tag{8}$$

The following equation is true for the high-pass filter since it's a mirror of the low-pass filter, that is $H_1(z) = H_0(-z)$,

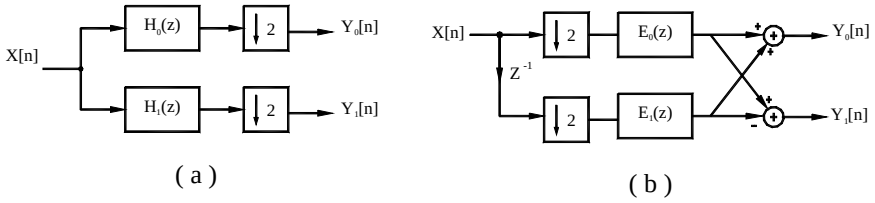


Fig. 3. Polyphase Realization of the Analysis Filter Bank : (a).Original Direct Structure; (b). Polyphase Structure.

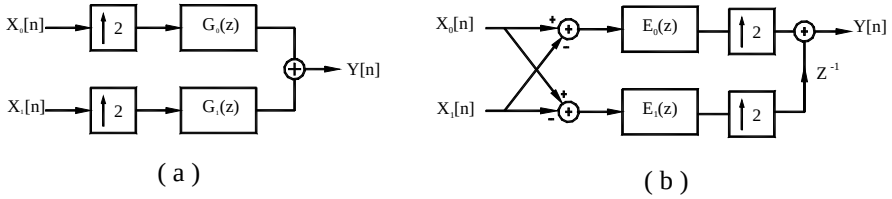


Fig. 4. Polyphase Realization of the Synthesis Filter Bank;(a).Original Direct Structure; (b). Polyphase Stucture.

$$H_1(z) = E_0(z^2) - z^{-1}E_1(z^2), \quad (9)$$

Using Equations (8) and (9), and making use of the multirate cascade equivalence (Vaidyanathan, 1993), we obtain the final polyphase structure of the analysis filter bank shown in Figure 3b. Analysis of the polyphase structure reveals that we have achieved a significant computation reduction (roughly a quarter of the computational complexity) in exchange for a modest increase in algorithm complexity and control.

3.1.2 Synthesis Filter Bank

The synthesis filter bank shown in Figure 4a represents the basic building block of the inverse discrete wavelet transform. It consists of two interpolators connected in parallel; the upper is a low pass filter proceeded by an up-sampler, and the lower is a high pass filter proceeded by an up-sampler. The up-sampler inserts an equidistant zero-valued sample between every two consecutive samples on the input sequence *x[n]* . An output sequence *y[n]* is developed such that y*[n] = x[n/2]* for even indices of *n,* and *0* otherwise. This makes the sampling rate of the output sequence *y[n]* twice as large as the sampling rate of the original sequence *x[n].*

We immediately observe a source of inefficiency in

this simple interpolation scheme. One out of every two samples presented to the filter represents the actual data sample, and the other sample is zero. Therefore, its clear computation power is being wasted in performing arithmetic operations on zero values. To avoid such multiply-by-zero arithmetic operations, a more efficient, but equivalent, implementation of the synthesis filter bank exists, and it is based on the concept of polyphase decomposition. If we expand $G_0(z)$ to its polyphase form as explained earlier, we get:

$$G_0(z) = E_0(z^2) + z^{-1}E_1(z^2), \tag{10}$$

The polyphase representation of the high-pass filter G_1 (z) follows directly:

$$G_1(z) = -E_0(z^2) + z^{-1}E_1(z^2), \tag{11}$$

Using Equations (10) and (11), and making use of the multirate cascade equivalence (Vaidyanathan, 1993), we obtain the polyphase structure of the synthesis filter bank shown in Figure 4b. Analysis of the polyphase structure reveals that we have achieved a significant computation reduction (roughly a quarter of the computational complexity) in exchange for a modest increase in algorithm complexity and control.

3.2 Distributed Arithmetic Sub-Filters

Distributed arithmetic is an efficient method for computing the inner product operation which constitutes the core of the discrete wavelet transform. Mathematical derivation of distributed arithmetic is extremely simple; a mix of Boolean and ordinary algebra (Mintzer). Let the variable Y hold the result of an inner product operation between a data vector x and a coefficient vector a. The distributed arithmetic representation the inner product operation is given as follows:

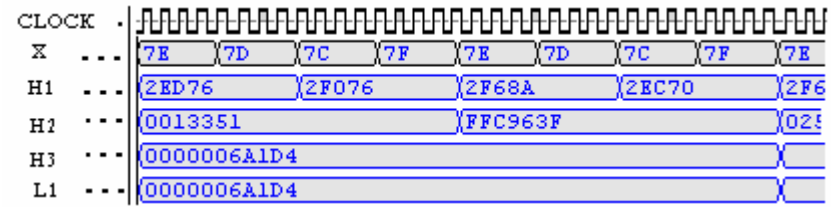
$$Y = \sum_{j=1}^{B-1}\left[\sum_{i=1}^{N}\chi_{ij}\,a_i\right]2^{-j} + \sum_{i-1}^{N}a_i\left(-\chi_{i0}\right) = \sum_{j=1}^{B-1}F_j\,2^{-j} - F_0 \tag{12}$$

Where the input data words x_i have been represented by the 2's complement number presentation in order to bound number growth under multiplication. The variable x_{ij} is the j^{th} bit of the x_i word which is Boolean, B is the number of bits of each input data word and x_{0i} is the sign bit. Distributed arithmetic is based on the observation that the function F_j can only take 2^N different values that can be pre-computed offline and stored in a look-up table. Bit j of each data x_{ij} is then used to address this look-up table. Equation (11) clearly shows that the only three different operations required for calculating the inner product. First, a look-up to obtain the value of F_j, then addition or subtraction, and finally a division by two that can be realized by a shift. FIR filters and the inner product operation described so far differ only in how they handle the input data.

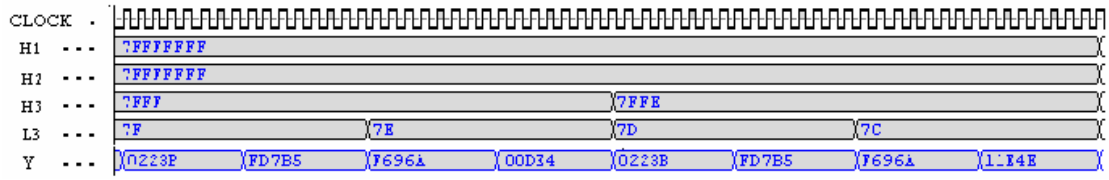
Each sub-filter in the polyphase DWT analysis and synthesis structures, described in the previous subsection, is implemented as a distributed arithmetic FIR filter consisting of a Look-Up Table (LUT) to store all possible partial products over the FIR filter coefficient of Table (1), a cascade of shift registers and a scaling accumulator, as shown in Figure 5a. Input samples are presented to the input parallel-to serial shift register at the input signal sample rate. As the input sample is serialized, the bit-wide output is presented to the bit-serial shift register cascade,1-bit at a time. The cascade stores the input sample history in a bit-serial format and is used in forming the required inner-product computation. The bit outputs of the shift register cascade are used as address inputs to the look-up table. Partial results from the look-up table are summed by the scaling accumulator to form a final result at the filter output port.

Since the LUT size in a distributed arithmetic implementation increases exponentially with the number of coefficients, the LUT access time can be a bottleneck for the speed of the whole system when the LUT size becomes large. Hence; we decomposed the 8-bit LUT shown in Figure 5a into two 4-bit LUTs, and added their outputs using a two-input accumulator. The modified partitioned-LUT architecture is shown in Figure 5b. The total size of storage is now reduced since the accumulator is less costly than the larger 8-bit LUT. Furthermore, partitioning the larger LUT into two smaller LUTs accessed in parallel reduces access time. In addition, throughput of the filter is maintained regardless of the length of the FIR filter. This

Fig. 5. Distributed Arithmetic Realizations of the FIR Filter: (a). Single-LUT Realization; (b) Efficient Partitioned-LUT Realization.



(a)



(b)

Fig. 6. Simplified Functional Verilog Simulation of the Discrete Wavelet Transform : (a). Forward DWT; (b). Inverse DWT.

Table 2: Throughput performance of different FPGA implementations.

Implementation	Throughput Performance (MHz)	
	Forward Discrete Wavelet Transform	Inverse Discrete Wavelet Transform
Conventional	*14.11*	*11.6*
Distributed Arithmetic	*26.0*	*23.7*
Polyphase Decomposition	*104.6*	*98.5*
Polyphase & Distributed Arithmetic	*131.7*	*119.6*

Table 3: Hardware utilization of different FPGA implementations.

Implementation	Utilization Performance (Slice)	
	Forward Discrete Wavelet Transform	Inverse Discrete Wavelet Transform
Conventional	*560 (18%)*	*619 (20%)*
Distributed Arithmetic	*374 (12%)*	*461 (15%)*
Polyphase Decomposition	*651 (21%)*	*708 (23%)*
Polyphase & Distributed Arithmetic	*830 (27%)*	*922 (30%)*

feature is particularly attractive for flexible implementations of different wavelet types since each type has a different set of filter coefficients.

4. Functional Simulation

Functional simulation is a major prerequisite step towards a correct and efficient FPGA implementation of the discrete wavelet transform. Therefore, the implementation described in the previous section, was modeled by the Verilog hardware description language and verified by its functional simulator (Bhasker, 1997) Simulation waveforms of the forward and inverse wavelet transforms are displayed in Figure 6. The waveforms prove that the implementation execute the operation of the wavelet transform correctly.

We used uniformly distributed 8-bit random input

samples to generate the simulation waveforms. We also maintained sufficient precision of the intermediate and output coefficients since allocating sufficient bits to the intermediate and output coefficients is a necessary step to keep the perfect reconstruction capabilities of the discrete wavelet transform. If we allocate fewer bits than necessary, the output of the inverse discrete wavelet transform will not be the same as a delayed version of the input of the forward discrete wavelet transform. Also, if we're dealing with an image compression application, the decompressed image will suffer from some defects, such as ringing effects and blurring artifacts.

Simulation waveform of the forward wavelet transform architecture of Figure 1a is illustrated in Figure 6a. As an input sample X enters the first filter bank stage at a rate of 1sample/ clock, one sample (H_1) leaves to the output, and another sample (L_1) leaves to the second stage, both at a rate of 1sample/ 2 clocks. Similarly, the second stage sends a sample to the output (H_2), and another sample (L_2) to the third stage, both at a rate of 1sample/ 4 clocks. Finally, the third stage generates two samples (L_3 and H_3) at a rate of 1 sample/ 8 clocks.

Simulation waveform of the inverse wavelet transform architecture of Figure 1b is illustrated in Figure 6b. The first filter bank stage receives two inputs (H_3 and L_3), both produced from the third stage of the forward DWT at a rate of 1sample/ 8 clocks. The stage up-samples each of them by a factor of 2, and sends out their filtered summation at the rate of 1sample/ 4 clocks to the second stage, to be processed with an input sample coming from the second stage of the forward DWT stage at a rate of 1sample/ 4 clocks (H_2). Similarly, the second stage up-samples both by a factor of 2, and then sends out their filtered summation at a rate of 1sample/ 2 clocks to the third stage to be processed with an input sample coming from the first stage of the forward DWT at a rate of 1sample/ 2 clocks (H_1). Finally, the third stage up-samples both by a factor of 2, and then sends out their filtered summation at a rate of 1sample/ clock to the output. This last output represents the reconstructed signal.

5. Discussion

In this section, we present performance results of the parallel polyphase and DA implementation described in section three. We also show how the results exceed considerably those obtained for other implementations of the transform.

5.1 Experimental Results

We carried out the implementation using a prototyping board called XSV-300 FPGA Board. The board is developed by XESS Inc. (2002), and is based on a single XCV300 FPGA chip (2000).This chip contains 3072 slices (322,970 gates), where each slice contains 4-input, 1-output LUTs and two registers. The LUTs allow any function of five inputs or two functions of four inputs to be created within a CLB slice. Furthermore, the chip can operate at a maximum clock speed of 200 MHz. Performance is evaluated with respect to two metrics; throughput (sample rate) and is given in terms of the clock speed, and device utilization, and is given in terms number of logic slices used by the implementation. Using these two metrics, the results of the polyphase and DA implementation are as follows. The forward discrete wavelet transform implementation operated at a throughput of 131.7 MHz, and required 830 Virtex slices which represents 27% of the total slices. Throughout the inverse discrete wavelet transform implementation was 119.6 MHz, and the hardware requirement was 922 slices which represents 30% of the total Virtex slices.

5.3 Performance Analysis

In what follows, we study the effects of using the polyphase decomposition and the distributed arithmetic techniques, separately. Therefore, we carried out three different implementations, and recorded their results in Tables (2 & 3). The first is a direct conventional implementation is in which all filters in the DWT tree were implemented using the conventional direct FIR structure shown in Figure 2a. The second is a polyphase implementation in which all filters in the DWT tree were implemented using the polyphase structure shown in Figure 3b. The third is a distributed arithmetic

implementation in which all filters in the DWT tree were implemented using the distributed arithmetic FIR structure shown in Figure 5b.

Referring to Table (2), it has been noticed that the throughput of the distributed arithmetic implementation is higher than the throughput of the direct conventional implementation. This is expected since the distributed arithmetic implementation replaced the time-consuming conventional multiply accumulate operations with fast look-up tables and shift operations. Furthermore, partial products of all multiply accumulate operations were pre-computed offline and stored in the LUTs, thus saving a great amount of real-time computation. As for Virtex slice utilization, Table (3) indicates that the distributed arithmetic implementation, uses less hardware resources than the direct implementation which uses conventional arithmetic. This is also expected since the conventional arithmetic multiplier requires more logic resources than the distributed arithmetic multiplier which requires small LUTs, simples adders and shift registers.

It is also noted from the results illustrated in Table (2), that the throughput of the polyphase implementation is much higher than the throughput of the direct implementation. This is expected since the polyphase implementation avoids unnecessary decimator and interpolator computations as explained in section 3. Furthermore, realizing the different filter banks of the transform in parallel contributed significantly to the reduction of the total computation time, and in turn to the considerable increase in the sample throughput. As for the hardware resources requirements of the two implementations, Table (3) indicates that the requirements are comparable, with the polyphase implementation requiring slightly more than the direct implementation. This is due to the fact that parallelizing sequential structures necessitates using more hardware resources.

Finally, the wavelet transform was implemented on the TMS320C6711; a Texas Instrument digital signal processing board with a complex architecture suitable for image processing applications (Kehtarnavaz Keramat, 2001). The board can operate at 150 MHz, with a peak performance of 900 MFLOPS (Texas Instruments corporation, 2000). Also, to complete the performance evaluation circle, we coded the transforms in C, and executed the corresponding programs on a conventional PC powered by an 800 MHz Pentium III processor. It is noted from the results illustrated in Figure 7, that all the FPGA implementations perform much better than the TMS20C6711 and Pentium III software implementations. The superior performance of the FPGA-based implementations is attributed to the highly parallel, pipelined and distributed architecture of Xilix Virtex FPGA. Moreover, it should be noted that the Virtex FPGAs offer more than high speed for many embedded applications. They offer compact implementation, low cost and low power consumption; things which can't be offered by any software implementation.

6. Concluding Remarks

In this paper, FPGA implementations of the discrete wavelet transform and its inverse were simulated and realized in a reconfigurable computing hardware board based on the advanced Xilinx Virtex FPGAs. According to the results obtained for the various implementations, we observed that the implementation which was based on the distributed arithmetic and ploy phase decomposition techniques achieved the best performance results. We also observed that the TMS320C6711 digital signal processor performed much better than the Pentium III , however, its performance is still much lower than the performance of the least efficient, direct FPGA implementation. One final remark is that the implementation is applicable to image-based applications where the image data is two dimensional. The 2-D transformation is straightforward, and can be easily achieved by inserting a matrix transpose module between two 1-D discrete wavelet transform modules. The 1-D discrete wavelet transform is first performed on each row of the 2-D image data matrix. This is followed by a matrix transposition operation. Next, the discrete wavelet transform is executed on each column of the matrix.

Fig. 7. Throughput Comparison Between Different DWT Implementations.

REFERENCES

Bellanger, M., Bonnerot, G. and Coudreuse, M. 1976. Digital Filtering by Polyphase Network: Application to Sample Rate Alteration and Filter Banks, *IEEE Acoustic Speech Signal Proc*., 24: 109-114.

Beylkin, G., Coifman, R. and Rokhlin, V. 1992. Wavelets in Numerical Analysis in *Wavelets and Their Applications*. New York: Jones and Bartlett, 181-210.

Bhasker, J. 1997. *A Verilog HDL Primer*. PA: Star Galaxy Publishing.

Burrus, C., Gopinath, R. and Guo, H. 1998. *Introduction to Wavelets and Wavelet Transforms.* New Jersey: Prentice Hall.

Chakabarti, C. and Vishwanath, M. 1995. Efficient Realizations of the Discrete and Continuous Wavelet Transforms: from Single Chip Implementations to Mappings on SIMD Array Computers, *IEEE Trans. Signal Processing*, 43 (3): 759-771.

Chakabarti, C., Vishwanath, M. and Owens, R. 1996. Architectures for Wavelet Transforms: A Survey, *Journal of VLSI Signal Processing*, 14 (2): 171-192.

Daubechies, I. 1988. Orthonomal Bases of Compactly Supported Wavelets, *Comm. Pure Appl. Math*, (41): 906-966.

Ebrahimi, T. and Pereira, F. *2002. The MPEG-4 Book*. Prentice Hall, July.

Kehtarnavaz, N. and Keramat, M. 2001. *DSP System Design Using the TMS320C6000*. New Jersey: Prentice Hall.

Knowles, G. 1990. VLSI Architecture for the Discrete Wavelet Transform, *Electron Letters*, 26 (15): 1184-1185.

Kollig, P., Al-Hashimi, B. and Abbot, K. 1997. FPGA Implementation of High Performance FIR Filters, In *Proc. International Symposium on Circuits and Systems*.

Mallat, S. A 1989. Theory for Multiresolution Signal Decomposition: The Wavelet Representation, *IEEE Trans. Pattern Anal. And Machine Intell*., 11 (7): 674-693.

Mintzer, L. The Role of Distributed Arithmetic in FPGAs, Xilinx Corporation.

Oppenheim, A. and Schafer, R. 1999. *Discrete Signal Processing*. New Jersy: Prentice Hall.

Parhi, K. and Nishitani, T. 1993. VLSI Architectures for Discrete Wavelet Transforms, *IEEE Trans. VLSI Systems*, 191-202.

Riol, O. and Vetterli, M. 1991. Wavelets and Signal Processing, *IEEE Signal Processing Magazine*, 8 (4):14-38.

Seals, R. and Whapshott, G. 1997. *Programmable Logic: PLDs and FPGAs*. UK: Macmillan.

Shand, M. 1995. Flexible Image Acquisition using Reconfigurable Hardware, In *Proc. of the IEEE Workshop on Filed Programmable Custom Computing Machines*, Napa, Ca, Apr.

Smith, M. 1997. *Application-Specific Integrated Circuits*. USA: Addison Wesley Longman.

Strang, G. and Nguyen, T. 1996. *Wavelets and Filter Banks*. MA: Wellesley-Cambridge Press.

Texas Instruments Corporation. 2000. *TMS320C6711 Data Sheet.*

Vaidyanathan, P. 1993. *Multirate Systems and Filter Banks*. New Jersey: Prentice Hall.

Villasenor, J., Schoner, B. and Jones, C. 1995. Video Communication using Rapidly Reconfigurable Hardware, *IEEE Transactions on Circuits and Systems for Video Technology,* 5 (12): 565-567.

White, S. 1989. Applications of Distributed Arithmetic to

Digital Signal Processing: A Tutorial, In *IEEE ASSP Magazine*, 4-19.

Xess Corporation. www.xess.com. 2002.

Xilinx Corporation. 1998. Xilinx Breaks One Million-Gate Barrier with Delivery of New Virtex Series.

Xilinx Corporation. *Virtex Data Sheet*, 2000.

تنفيذ محولة الموجيات المنفصلة باستخدام رقائق السيلكون القابلة للتشكيل مع الحساب الموزع والتحليل الزاوي

علي الحاج*

ملخص

بات من الضروري تنفيذ محولة الموجيات المنفصلة بواسطة رقائق السيلكون المتكاملة الموجهة للاستخدامات المتخصصة (ASICs)، وذلك لما تحتاجه هذه المحولة من كم هائل من الحسابات. في هذا البحث نقدم طريقة فائقة السرعة لتنفيذ المحولة باستخدام نوع من رقائق السيلكون المتكاملة الموجهة للاستخدامات المتخصصة (ASICs)، وهذا النوع يعـرف برقائق السيلكـون المتكاملة القابلة للتشكيـل (FPGAs). تعتمد الطريقة على اعادة صياغة المحولة رياضيا باستخدام الحساب الموزع والتحليل الزاوي ذي الاطوار المتعددة. وتتفق الصياغة الجديدة للمحولة مع تركيبة رقائق FPGA المستخدمة، وهذا يعني استغلالاً كاملاً لمكونات الرقائق وبالتالي اعطاء سرعات عالية. الطريقة المقترحة لها مزايا مثل السهولة في التنفيذ، واستخدام المحولة باحجام مختلفة، والقدرة على تنفيذ المحولة في الاتجاهين الأمامي والعكسي. وأخيرا، نعرض في نهاية البحث نتائجنا التي تثبت قدرة رقائق FPGA مع الحساب الموزع و التحليل الزاوي على تسريع تنفيذ محولة الموجيات المنفصلة، ونقارن النتائج مع نتائج حصلنا عليها عند تنفيذ المحولة باستخدام ادوات تسريع اخرى.

* جامعة الأميرة سمية، عمان، الأردن. تاريخ استلام البحث 2003/11/21، وتاريخ قبوله 2004/7/5.