International Journal of Applied Science and Engineering
2003. 1, 2: 160-171

Fast Discrete Wavelet Transformation Using FPGAs and Distributed Arithmetic

Ali M. Al-Haj

Department of Electronics Engineering,
Princess Sumaya University for Technology,
Al-Jubeiha P.O. Box 1438, Amman 11941, Jordan.

Abstract: The discrete wavelet transform has gained the reputation of being a very effective signal analysis tool for many practical applications. However, due to its computation-intensive nature, current implementations of the transform fall short of meeting real-time processing requirements of most applications. This paper describes a parallel implementation of the discrete wavelet transform and its inverse using high-density field programmable logic devices (FPGAs). The implementation exploits the lookup table-based architecture of Virtex FPGAs, by reformulating the wavelet computation in accordance with the distributed arithmetic algorithm. Performance results show that the distributed arithmetic formulation results in a considerable performance gain compared with the conventional arithmetic formulation of the wavelet computation. Finally, we show that the FPGA implementation outperforms alternative software implementations of the discrete wavelet transform.

Keywords: discrete wavelet transform; distributed arithmetic; FPGA implementation.

1. Introduction

The wavelet transform is an emerging signal processing technique that can be used to represent real-life non-stationary signals with high efficiency. Indeed, the wavelet transform is gaining momentum to become an alternative tool to traditional time-frequency representation techniques such as the discrete Fourier transform and the discrete cosine transform. By virtue of its multi-resolution representation capability, the wavelet transform has been used effectively in vital applications such as transient signal analysis [1], numerical analysis [2], computer vision [3], image compression [4], among many other audio-visual applications.

The discrete wavelet transform is computationally intensive and operates on large data sets. This factor, coupled with the demand for real time operation in many image processing tasks, made the traditional sequential computers fall short in meeting such requirements. In turn, this necessitated the search for high performance implementations at a reasonable cost. Implementations of the discrete wavelet transform can be grouped into two major categories; software implementations using programmable parallel systems, and dedicated hardware implementations using customized VLSI devices. Each implementation category presents different trade-offs in terms of performance, cost, power, and flexibility.

Several parallel systems that meet the computational requirements of the wavelet transform have been proposed [5, 6]. However, programming such multiprocessor systems is a tedious, difficult, and time consuming task.

Corresponding e-mail: ali@psut.edu.jo
Accepted for Publication: July 22, 2003

Moreover, multiprocessor implementations of the discrete wavelet transform are not cost effective since parallelism comes at the expense of augmenting the system with more processing engines operating in parallel. This is in addition to the fact that the discrete wavelet transform is mostly needed to be embedded in consumer electronics, and thus a single chip hardware implementation is more desirable than a multi-chip parallel system implementation.

Several VLSI architectures have been proposed for the implementation of the discrete wavelet transform. The first architecture, presented by Knowles [7], uses many large multiplexers for storing intermediate results. Parhi and Nishitani proposed a folded architecture that has shorter latency [8], however, it requires complex routing and control network. Chakabarti [9] proposed a systolic architecture, but also it requires many parallel hardware and complex routing. In general, custom VLSI circuits are inherently inflexible and their development is costly and time consuming, and thus they are not an attractive option for implementing the wavelet transform.

Filed programmable gate arrays (FPGAs) provide a new implementation platform for the discrete wavelet transform. FPGAs maintain the advantages of the custom functionality of VLSI ASIC devices, while avoiding the high development costs and the inability to make design modifications after production [10]. Furthermore, FPGAs inherit design flexibility and adaptability of software implementations. In this paper we describe a parallel and high speed implementation of the discrete wavelet transform and its inverse using Virtex FPGAs produced by Xilinx [11].

We make maximal utilization of the look-up table (LUT) architecture of Virtex FPGAs by reformulating the wavelet transform computation in accordance with the distributed arithmetic algorithm [12]. Distributed arithmetic makes extensive use of look-up tables , which makes it ideal for implementing the discrete wavelet transform functions onto the LUT-based architecture of Virtex FPGAs. Moreover, distributed arithmetic is suitable for low power portable applications because it allows replacement of costly multipliers with shifts and look-up tables.

Indeed, one of the unique features of our discrete wavelet transform implementation is exploiting the natural match between the Virtex architecture and distributed arithmetic. Three more unique features are worth mentioning at this point. The first is the flexibility of the implementation which is made possible by virtue of the re-programmability of FPGAs which allows easy modification of wavelet type. The second is that, unlike most reported implementations which concentrate on architecture development, this implementation goes down to the actual implementation level. Finally, this paper describes implementations for both the forward and inverse transforms, whereas most papers report on the implementation of the forward wavelet transform only.

The paper is organized as follows. Section two gives an introduction to basic wavelets computation. Section three highlights the architectural match between filed programmable gate arrays and distributed arithmetic. Section four describes the implementation of discreet wavelet transform and its inverse using the distributed arithmetic method. Section five describes functional simulation of the forward and inverse implementations. Section six presents the performance results and compares them with the performance results obtained for alternative FPGA and software implementations. Finally, section seven presents some concluding remarks.

2. Basic wavelet computation

Wavelets are special functions which, in a form analogous to sines and cosines in Fourier analysis, are used as basal functions for representing signals. They provide powerful multiresolution tool for the analysis of non-stationary signals with good time localization

information [13]. 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 [14]. Based on this algorithm, the 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, and m is the summation running index of the analysis filters' coefficients, respectively. Eq. (1) can then be used for obtaining the coefficients of subsequent stages. In practice this decomposition is performed only for a few stages.

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 follows:

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

where $g_0(n)$ and $g_1(n)$ are respectively the low-pass and high-pass synthesis filters corresponding to the mother wavelet, and and l is the summation running index of the analysis filters' coefficients. 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 [15]. The filters coefficients corresponding to this wavelet type are shown in Table 1, and have been taken from [16]. H_0 and H_1 are the input decomposition filters and G_0 and G_1 are the output reconstruction filters.

Table 1. Daubechies 8-tap wavelet coefficients

H_0	H_1	G_0	G_1
-0.0106	0.2304	-0.2304	-0.0106
-0.0329	0.7148	0.7148	0.0329
0.0308	0.6309	-0.6309	0.0308
0.1870	-0.0280	-0.0280	-0.187
-0.0280	-0.1870	0.1870	-0.0280
-0.6309	0.0308	0.0329	0.6309
0.7148	0.0329	-0.0329	0.7148
-0.2304	-0.0106	-0.0106	0.2304

3. Distributed arithmetic & virtex FPGAs

Distributed arithmetic is an efficient method for computing the inner product operation which constitutes the core of the discrete wavelet transform. In this section we briefly describe the mathematical derivation of the distributed arithmetic algorithm. We follow the derivation with an overview of Xilinx's Virtex FPGA architecture and a description of show how it matches the lookup table structure of distributed arithmetic computation .

3.1. Distributed arithmetic

Mathematical derivation of distributed arithmetic is extremely simple; a mix of Boolean and ordinary algebra [17]. Let the variable Y hold the result of an inner product operation between a data vector x and a coefficient vector a. The conventional representation the inner product operation is given as follows:

$$Y = \sum_{i=0}^{N-1} a_i x_i = \sum_{i=1}^{N} a_i \left[-x_{i0} + \sum_{j=1}^{B-1} x_{ij} 2^{-j} \right] \quad (4)$$

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. Interchange the order of summation of Eq. (4), we get:

$$Y = \sum_{j=1}^{B-1}\left[\sum_{i=1}^{N} x_{ij} a_i\right] 2^{-j} + \sum_{i=1}^{N} a_i(-x_{i0}) = \sum_{j=1}^{B-1} F_j 2^{-j} - F \quad (5)$$

Distributed arithmetic is based on the observation that the function F_j can only take 2^N different values that can be pre-computed off-line and stored in a look-up table. Bit j of each data x_{ij} is then used to address this look-up table. Eq. (5) 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.

In its most obvious and direct form, distributed arithmetic computations are bit-serial in nature, i.e., each bit of the input samples must be indexed in turn before a new output sample becomes available. When the input samples are represented with B bits of precision, B clock cycles are required to complete an inner-product calculation. An example of a distributed arithmetic implementation of a 4-element inner product operation is shown in Figure 1 along with the conventional implementation of the same product operation.

3.2. Virtex FPGAs

One of most advanced FPGA families in industry is the FPGA series produced by Xilinx [18]. The Virtex user-programmable gate array comprises two major configurable elements: configurable logic blocks (CLBs) and input/output blocks (IOBs). CLBs provide the functional elements for constructing logic and IOBs provide the interface between the package pins and the CLBs.

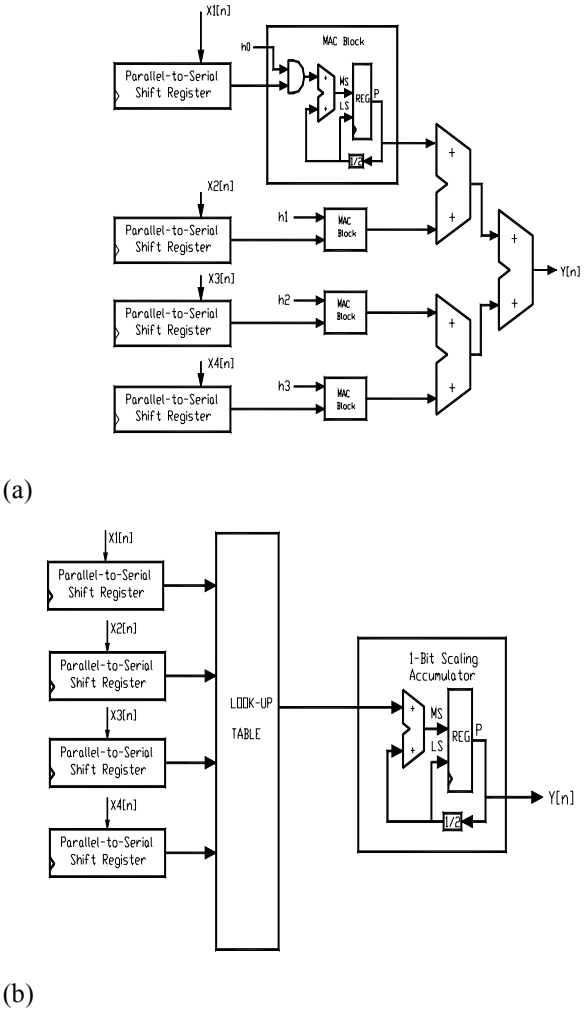

Figure 1. Multiply accumulate operation (a) conventional implementation and (b) distributed Arithmetic implementation

3.2.1. Internal configuration

The basic Virtex logic element in a CLB is the slice [19]. Two slices are present in each CLB as shown in Figure 2. Each slice contains 4-input, 1-output LUTs and two registers. Interconnections between these elements are configured by multiplexers controlled by SRAM cells programmed by a user's bitstream. The LUTs allow any function of five inputs, and two functions of four inputs, or

some functions of up to nine inputs to be created within a CLB slice. The outputs of theses functions may be registered, or the registers may be used independently of the LUTs. This structure allows a very powerful method of implementing arbitrary, complex digital logic.

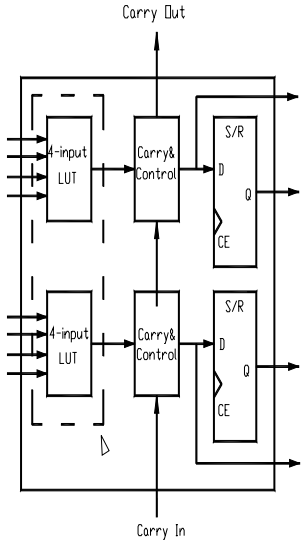

Figure 2. Simplified Virtex configurable slice

3.2.2. Look-up table implementation

Virtex slices have the ability to implement distributed memory instead of logic. Each 4-input LUT in a slice may be used to implement a 16x1 ROM or RAM, or the two LUTs may be combined together to create a 32x1 ROM or RAM or a 16x1 dual-port RAM. This allows each slice to trade logic resources for memory in order to maximize the resources available for a particular application.

Distributed Arithmetic for inner product generation can be easily implemented in the LUT-based Xilinx Virtex FPGAs. The inner product production basically consists of table-lookup operations and additions. Thus RAM or ROM can be employed holding table values, and table lookup operations can be performed, and then a parallel adder usually follows to sum up LUT values provided by ROM or RAMs.

4. Distributed arithmetic implementation

The discrete wavelet transform equations described in the previous section can be efficiently computed using the quadratic mirror filter (QMF) tree shown in Figure 3. In this section we describe a distributed arithmetic implementation of the QMF tree. The implementation starts by deriving the distributed arithmetic structure of a single FIR filter, and then by describing the implementation of the QMF filter banks of both the forward and discrete wavelet transforms.

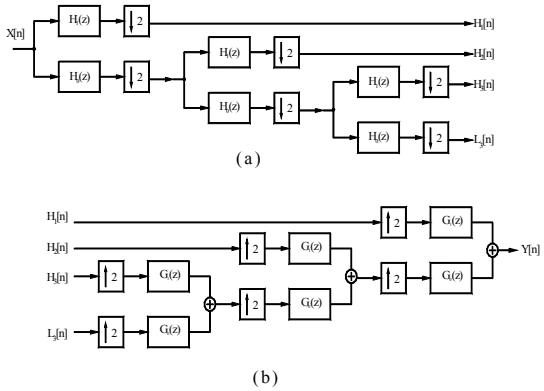

Figure 3. Mallat's quadratic mirror Filter tree (a) DWT and (b) Inverse DWT tree

4.1. Distributed arithmetic FIR filter

All filters in the pyramid tree structure shown in Figure 3 are constructed using FIR filters because of their inherent stability. A casual FIR filter of length ***M*** is characterized by:

$$H(z) = \sum_{k=0}^{M-1} h[k]\, z^{-k} \tag{6}$$

Most discrete wavelet transform implementations reported in literature employ the direct form structure shown in Figure 4. As shown in the figure, each filter tap consists of a delay element, an adder, and a multiplier [20]. However, a major drawback of this implementation is that filter throughput is inversely

proportional to the number of filter taps. That is, as filter length is increased, the filter throughput is proportionately decreased.

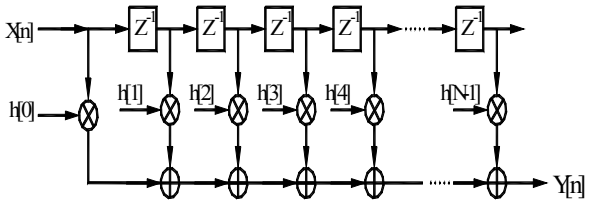

Figure 4. Direct FIR filter structure

Distributed arithmetic implementation of an FIR filter consists of a look-up table (LUT), a cascade of shift registers and a scaling accumulator, as shown in Figure 5.

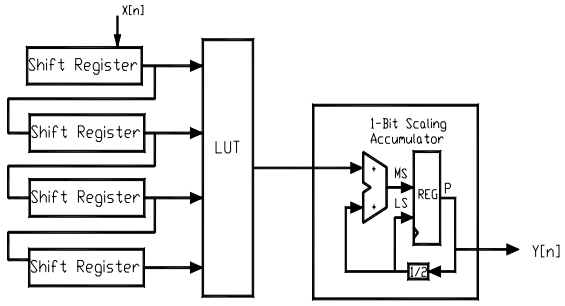

Figure 5. Distributed arithmetic FIR filter

The LUT stores all possible partial products over the FIR filter coefficient space given in Table 1. 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 5 into two 4-bit LUTs, and added their outputs using a two-input accumulator. The modified partitioned-LUT architecture is shown in Figure 6.

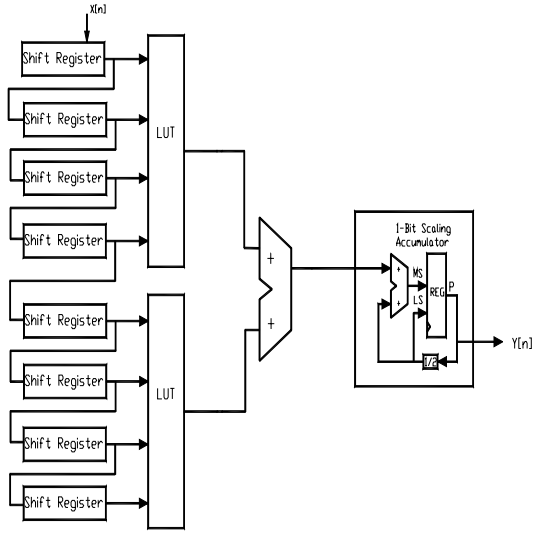

Figure 6. Efficient distributed arithmetic implementation of the FIR filter

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

4.2. Forward DWT implementation

The basic building block of the forward discrete wavelet transform filter bank is the decimator which consists of an FIR filter followed by a down-sampling operator [21]. Down-sampling an input sequence *x[n]* by an integer value of 2, consists of generating an

output sequence *y[n]* according to the relation *y[n] = x[2n]*. Accordingly, the sequence *y[n]* has a sampling rate equal to half of that of *x[n]*.

We implemented the decimator as shown in Figure 7a. An active-high output control pin, labeled DATA RDY, has been implemented in the distributed arithmetic FIR structure and connected directly to the CLK input of a 1-bit counter. The input port of the FIR filter is connected to the input samples source, whereas the output port is connected to a parallel-load register. The register loads its input bits in parallel upon receiving a high signal on its CLK input from the counter, and blocks its input otherwise.

Assuming unsigned 8-bit input samples, the decimator operates as follows. When the DATA RDY signal is activated, every time the FIR completes a filter operation, it triggers the counter to advance to the next state. If the new state is 1, the parallel-load register is activated, and it stores the data received at its input from the FIR filter. If the new state is 0, the register is disabled, and consequently the FIR output is blocked from entering the register, and ultimately discarded. The above procedure repeats, so that when the counter has 1 on its output, the FIR data is stored , and when it has a 0 on its output, the FIR data is discarded.

4.3. Inverse DWT implementation

The basic building block of the inverse discrete wavelet transform filter bank is the interpolator which consists of an FIR filter proceeded by an up-sampling operator [21]. The up-sampler inserts an equidistant zero-valued sample between every two consecutive samples on the input sequence *x[n]* to develop an output sequence *y[n]* such that y*[n] = x[n/2]* for even indices of *n,* and *0* otherwise. The sampling rate of the output sequence *y[n]* is thus twice as large as the sampling rate of the original sequence *x[n]*.

We implemented the interpolation filter as shown in Figure 7b. The input port of the FIR filter is connected to the output port of a parallel-load register; whereas the input port of the register is connected directly to the input samples source. The operation of the register depends on the signal received on its active-high CLR (clear) input from the most significant output bit of a 4-bit counter.

Assuming the input samples source sends out successive samples separated by 16 clock periods, the interpolator operates as follows. Let an input sample be transferred, through the parallel-load register, to the FIR filter. The transfer process takes place during the first eight counts of the 4-bit counter in which the counter's MSB remains 0, thus enabling the register to transfer its input data to its output port.

During the next eight counts, the MSB of the count becomes 1, and thus clearing the register and consequently transferring zeros to its output. The zero output is maintained until the last count (FFFF H) is reached. The above procedure repeats so that an input sample enters the FIR filter during the first eight clocks, followed by a zero during the next eight clocks, and so on.

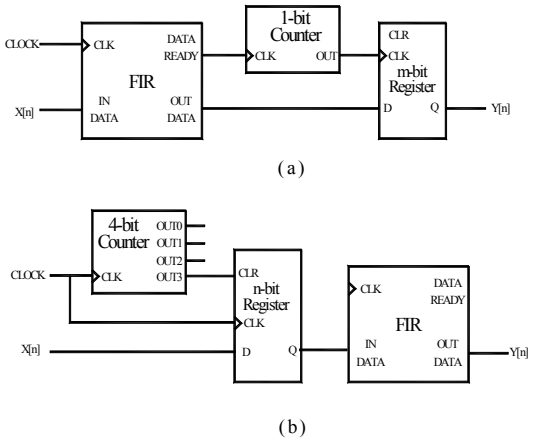

Figure 7. Implementation of the basic blocks of the (a) decimator block and (b) interpolator block

5. Functional simulation

Functional simulation is a major prerequisite step towards a correct and efficient FPGA implementation of the discrete wavelet transform. Therefore, the distributed arithmetic implementations, described in the previous section, were modeled by the Verilog hardware description language and verified by its functional simulator [22]. Simulation waveforms of the forward and inverse wavelet transforms are displayed in Figure 8. The waveforms prove that the implementations 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 form some defects, such as ringing effects and blurring artifacts.
Simulation waveform of the forward wavelet transform architecture of Figure 3a is illustrated in Figure 8.

As an input sample X enters the first filter bank stage at a rate of 1sample/8 clocks, one sample (H_1) leaves to the output, and another sample (L_1) leaves to the second stage, both at a rate of 1sample/ 16clocks. 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/32 clocks. Finally, the third stage generates two samples (L_3 and H_3) at a rate of 1 sample/ 64clocks. Simulation waveform of the inverse wavelet transform architecture of Figure 3b is illustrated in Figure 9. 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/64clock.

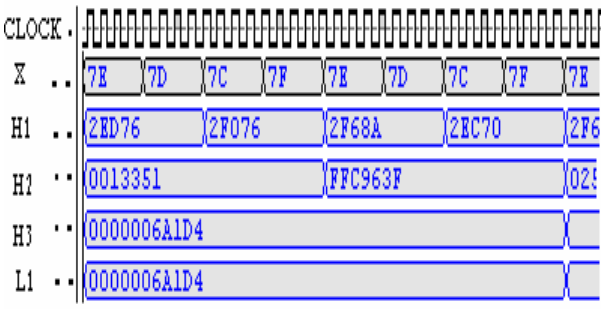

Figure 8. Simplified functional Verilog simulation of the forward DWT

The stage up-samples each of them by a factor of 2, and sends out their filtered summation at the rate of 1sample/32 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/32 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/16 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/ 16clocks (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/ 8clocks to the output. This last output represents the reconstructed signal.

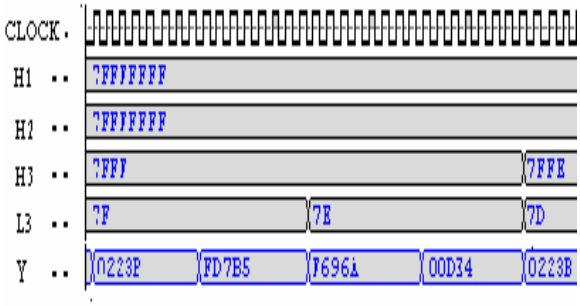

Figure 9. Simplified functional Verilog simulation of the inverse DWT

6. Performance evaluation

In this section we present the experimental results obtained for the distributed arithmetic implementation of the discrete wavelet transform and its inverse. We also compare the results with those of an FPGA implementation based on conventional arithmetic, and with the results of two software implementations.

6.1. Experimental results

We have implemented the parallel designs described in the previous section using one of the largest available Xilinx Virtex FPGA devices, XCV300. This device contains 322,970 gates (3072 slices) and can operate at a maximum clock speed of 200 MHz. Therefore, performance is usually measured with respect to two evaluation metrics; the throughput (sample rate) and is given in terms of the clock speed, and device utilization, and is given in terms number of Virtex logic slices used by the implementation.

The distributed arithmetic implementation was verified with Verilog HDL Simulator, and synthesized using Xilinx Foundation Series. The forward discrete wavelet transform implementation operated at a throughput of 92.7 MHz, and required 374 Virtex slices which represents around 12 % of the total 3072 slices. Throughout of the inverse discrete wavelet transform implementation was 89.1 MHz, and the hardware requirement was 461 slices which represents around 15 % of the total Virtex slices.

The bit stream corresponding to the implementation was downloaded to a prototyping board called the XSV-300 FPGA Board, developed by XESS Inc [23]. The board is based on a single Xilinx XCV300 FPGA. It can accept video with up to 9-bits of resolution and output video images through a 110 MHz, 24-bit RAMDAC. Two independent banks of 512K x 16 SRAM are provided for local buffering of signals and data.

6.2. Performance comparison

We implemented the discrete wavelet transform architecture shown in Figure 3 using the conventional arithmetic approach. The forward discrete wavelet transform achieved a throughput of 54.3 MHz, and required 560 Virtex slices which represents 18 % of the total Virtex slices, and the inverse discrete wavelet transform achieved a throughput was 47.8 MHz, and required 619 slices which represents around 20 % of the total Virtex slices.

It is noted from the results obtained above, and further illustrated in Figure 10, that the throughput of the distributed arithmetic implementation is higher than the throughput of the conventional arithmetic 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 precomputed offline and stored in the LUTs, thus saving a great a mount of real-time computation. As for Virtex slice utilization, distributed arithmetic, uses less hardware resources than the conventional arithmetic, as illustrated in Figure 11.

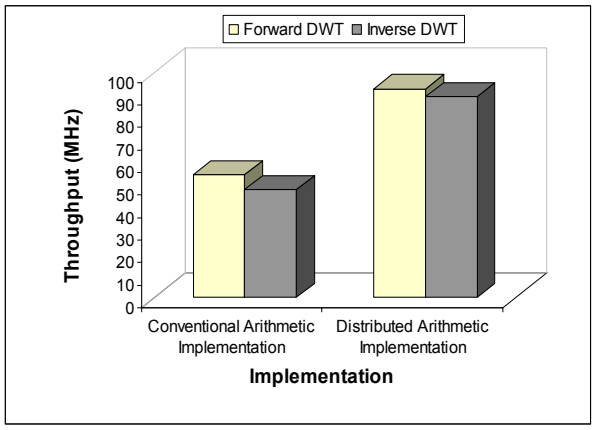

Figure 10. Comparison between the throughput of two DWT implementations

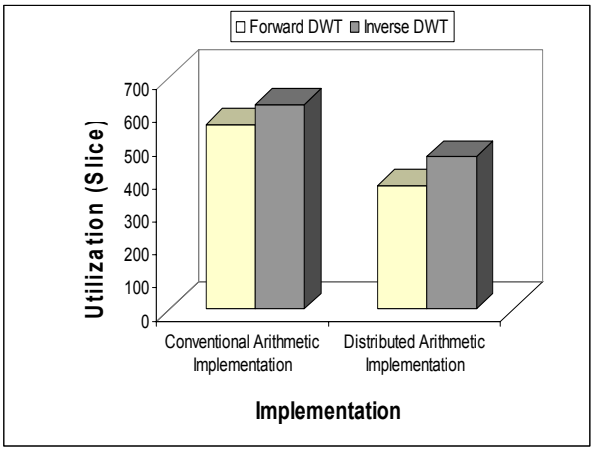

Figure 11. Comparison between the utilization of two DWT implementations

This is also expected since the conventional arithmetic multiplier requires much more logic resources than the distributed arithmetic multiplier which requires small LUTs, simples adders and shift registers. The wavelet transform was also implement- ed on the TMS320C6711; a Texas Instrument digital signal processor with an a complex architecture suitable for image processing applications [24]. The TMS320C6711 is a highly integrated single chip processor and can operate at 150 MHz (6.7 ns clock cycle with a peak performance of 900 MFLOPS [25].

The processor was programmed such that the main portion of the wavelet transform was written in C, and certain sections in assembly. Also, parallel instructions were used whenever possible to exploit the abundant parallelism inherent in the wavelet transform. Sample execution times obtained for both the forward and inverse discrete wavelet transforms were 0.153 μs (6.53 MHz) and 0.276 μs (3.62 MHz), respectively.

To complete the performance evaluation circle, we coded the forward and inverse wavelet transforms in C, and executed corresponding programs on a conventional PC powered by an 800 MHz Pentium III processor. The execution times obtained for both the forward and inverse discrete wavelet transforms were 0.1280 msec (0.00781 MHz) and 0.1485 msec (0.00673 MHz), respectively.

It is noted from the results obtained above, and summarized in Table 2, that the two 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 Xilinx 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.

Table 2. Throughput of different implementations

Implementation	Forward DWT (MHz)	Inverse DWT (MHz)
Pentium III	*0.00781*	*0.00673*
TMS320C6711	*6.530*	*3.620*
Conventional arithmetic	*54.3*	*47.8*
Distributed arithmetic	*92.7*	*89.1*

6.3. Ongoing research

After completing this FPGA implementation of the discrete wavelet transform and its inverse, we are now working on integrating a whole wavelet-based image compression system on a single, dynamic, runtime reconfigurable FPGA. A typical image compression system consists of an encoder and a decoder. At the encoder side, an image is first transformed to the frequency domain using the forward discrete wavelet transform.

The non-negligible wavelet coefficients are then quantized, and finally encoded using an appropriate entropy encoder. The decoder side reverses the whole encoding procedure described above. Transforming the 2-D image data can be done simply by inserting a matrix

transpose module between two 1-D discrete wavelet transform modules such as those described in this paper.

7. Conclusions

In this paper we reported on the performance of several implementations of the discrete wavelet transform and its inverse; two implementations using the highly parallel Virtex filed programmable gate array devices (FPGAs), and two software implementations; one using the TMS320C6711 digital signal processor and the other using the 800 MHz Pentium III Intel processor.

Based on the results obtained of the various implementations, we observed that the implementation which was based on the distributed arithmetic algorithm achieved the best performance results. This has been possible by reformulating the computation of the wavelet transform so that it matches the lookup architecture of the Viretx FPGA. It was also observed that the two software implementations were far inferior to the FPGA implementations in terms of execution speed. The TMS320C6711 digital signal processor performed much better than the Pentium III , however, its performance is still much lower the performance of the least efficient, direct FPGA implementation.

Finally, it may be concluded that using FPGAs, coupled with reformulating the computation of the wavelet transform in accordance with the distributed arithmetic algorithm, results in the performance levels required for real-time implementations.

References

[1] Riol, O. and Vetterli, M. 1991. Wavelets and signal processing. *IEEE Signal Processing Magazine,* 8, 4: 14-38.

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

[3] Field, D. J. 1999. Wavelets, vision and the statistics of natural scenes. *Philosophical Transactions of the Royal Society: Mathematical, Physical and Engineering Sciences,* 357, 1760: 2527-2542.

[4] Antonini, M., Barlaud, M., Mathieu, P., and Daubechies, I. 1992. Image coding using wavelet transform. *IEEE Transactions on Image Processing*, 1, 2: 205-220.

[5] Sava, H., Fleury, M., Downton, A., and Clark, A. 1997. Parallel pipeline implementation of wavelet transforms. *IEE Proceedings-Vision Image and Signal Processing*, 144: 6.

[6] Aware, Inc. 1991. "*Aware Wavelet Transform Processor (WTP) Preliminar*". Cambridge, MA.

[7] Knowles, G. 1990. VLSI architecture for the discrete wavelet transform. *Electron Letters,* 26, 15: 1184-1185.

[8] Parhi, K. and Nishitani, T. 1993. VLSI architectures for discrete wavelet transforms. *IEEE Transactions on VLSI Systems*: 191-202.

[9] 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 Transactions on Signal Processing*, 43, 3: 759-771.

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

[11] Xilinx Corporartion. 2002. www.xilinx.com.

[12] White, S. 1989. Applications of distributed arithmetic to digital signal processing: a tutorial. *IEEE ASSP Magazine*: 4-19.

[13] Burrus, C., Gopinath, R., and Guo, H. 1998. "*Introduction to Wavelets and Wavelet Transforms: A Primer"*. New Jersey: Prentice Hall.

[14] Mallat, S. 1989. A theory for multresolution signal decomposition: the wavelet representation. *IEEE Transactions on*

Pattern Analysis and Machine Intelligence, 11, 7: 674-693.

[15] Daubechies, I. 1988. Orthonomal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 41: 906-966.

[16] Salomon, D. 1999. "*Data Compression: The Complete Reference"*. Springer.

[17] Mintzer, L. 1996. *"The Role of Distributed Arithmetic in FPGAs"*. Xilinx Corporation.

[18] Xilinx Corporation. 1998. *"Xilinx breaks one million-gate barrier with delivery of new Virtex series"*.

[19] Xilinx Corporation. 2000. "*Virtex Data Sheet"*.

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

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

[22] Palnitkar, S. 1996. "*Verilog HDL"*. SunSoft Press.

[23] Xess Corporation. 2002. www.xess.com.

[24] Texas Instruments Corporation. 2000. "*TMS320C6711 Data Sheet"*.

[25] Kehtarnavaz N. and Simsek, B. 2000. "*C6x-Based Digital Signal Processing"*. New Jersey: Prentice Hall.