

A Novel Approach for Two Dimensional Discrete Wavelet Transform on FPGA

¹Alisha, ²Suresh Badugu

^{1,2}Dept. of ECE, K L University, Vaddeswaram, Andhra Pradesh, India

Abstract

Field 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. 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.

Keywords

Discrete Wavelet Transform, FPGA, DWT, STFT

I. Introduction

A wave is periodic and is a oscillating function of time or space. In contrast, wavelets are localized waves. They have their energy concentrated in time or space and are suited to analysis of transient signals. While Fourier Transform and STFT use waves to analyze signals, the Wavelet Transform uses wavelets of finite energy

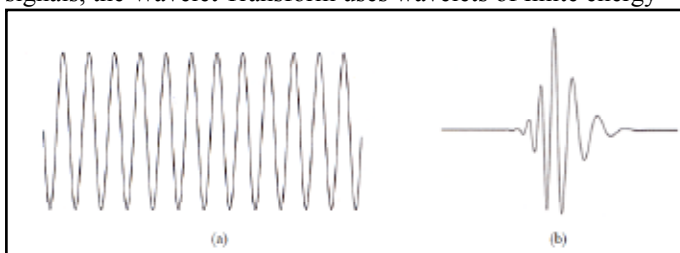


Fig. 1:(a) Wave, (b) Wavelet

The wavelet analysis is done similar to the STFT analysis. The signal to be analyzed is multiplied with a wavelet function just as it is multiplied with a window function in STFT, and then the transform is computed for each segment generated. However, unlike STFT, in Wavelet Transform, the width of the wavelet function changes with each spectral component. The Wavelet Transform, at high frequencies, gives good time resolution and poor frequency resolution, while at low frequencies; the Wavelet Transform gives good frequency resolution and poor time resolution.

B. The Discrete Wavelet Transform

The Discrete Wavelet Transform (DWT), which is based on sub-band coding, is found to yield a fast computation of Wavelet Transform. It is easy to implement and reduces the computation time and resources required.

The foundations of DWT go back to 1976 when techniques to decompose discrete time signals were devised. Similar work was done in speech signal coding which was named as sub-band coding. In 1983, a technique similar to sub-band coding was developed which was named pyramidal coding. Later many improvements were made to these coding schemes which resulted in efficient multi-resolution analysis schemes.

The fig. 2 shows a 2 level wavelet transform. For the low-pass and high-pass filters, I have used the same as that of JPEG2000, lossy transform, so that the same type of wavelet used in JPEG2000

format, it is quite hard to show a wavelet transform properly, because of the different ranges of different parts. I have chosen following things.

Firstly, I wanted to use a picture with flat surfaces, sharp edges and details. The standard image used for this purpose is Lenna, but I didn't want to use it because it is not in the public domain. I had a look at the big featured picture gallery con Commons and I have chosen Image: Lichtenstein img, processing test.png because it has lots of details at the bottom left rather than the top right. Moreover, the edges of the castle are really sharp, so they can be clearly seen in the wavelet transform.

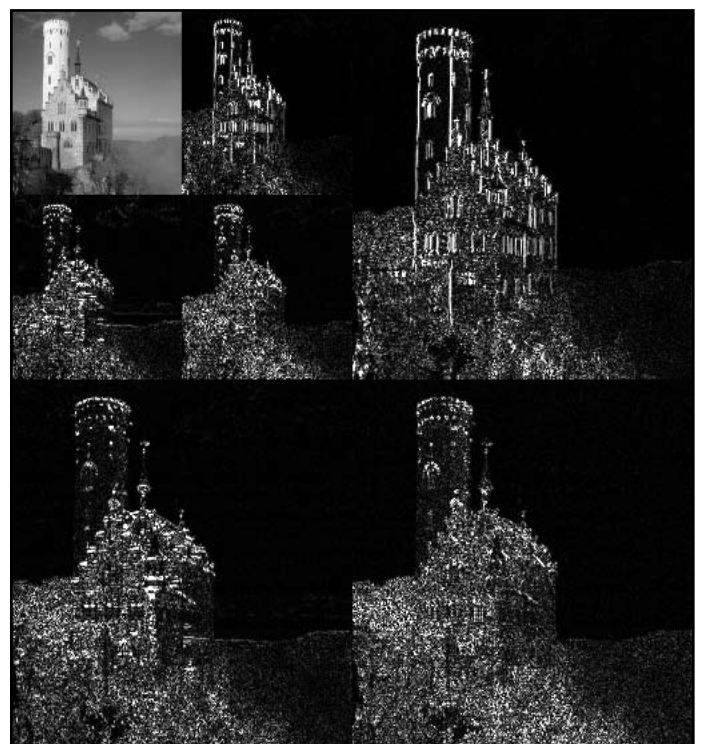


Fig. 2: 2D Discrete Wavelet Transform

B. A Field- Programmable Gate Array (FPGA)

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing—hence “field-programmable”. The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC) (circuit diagrams were previously used to specify the configuration, as they were for ASICs, but this is increasingly rare). Contemporary FPGAs have large resources of logic gates and RAM blocks to implement complex digital computations. As FPGA designs employ very fast I/Os and bidirectional data buses it becomes a challenge to verify correct timing of valid data within setup time and hold time. Floor planning enables resources allocation within FPGA to meet these time constraints. FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping, partial re-configuration of a portion of the design and the low non-recurring engineering costs relative to

an ASIC design (notwithstanding the generally higher unit cost), offer advantages for many applications.

FPGAs contain programmable logic components called “logic blocks”, and a hierarchy of reconfigurable interconnects that allow the blocks to be “wired together”—somewhat like many (changeable) logic gates that can be inter-wired in (many) different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

II. FPGA Architecture

The architecture of a field-programmable gate array (FPGA), as illustrated in fig. 1, is similar to that of a mask-programmable gate array (MPGA), consisting of an array of logic blocks that can be programmable interconnected to realize different designs. The major difference between FPGA's and MPGA's is that an MPGA is programmed using integrated circuit fabrication to form metal interconnections, while an FPGA is programmed via electrically programmable switches much the same as traditional programmable logic devices (PLD's). FPGA's can achieve much higher levels of integration than PLD's, however, due to their more complex routing architectures and logic implementations. PLD routing architectures are very simple but highly inefficient crossbar-like structures in which every output is directly connectable to every input through one switch. FPGA routing architectures provide a more efficient MPGA-like routing where each connection typically passes through several switches. In a PDL, logic is implemented using predominantly two-level AND-OR logic with wide input AND gates. In an FPGA logic is implemented using multiple levels of lower fanin gates, which is often much more compact than two-level implementations.

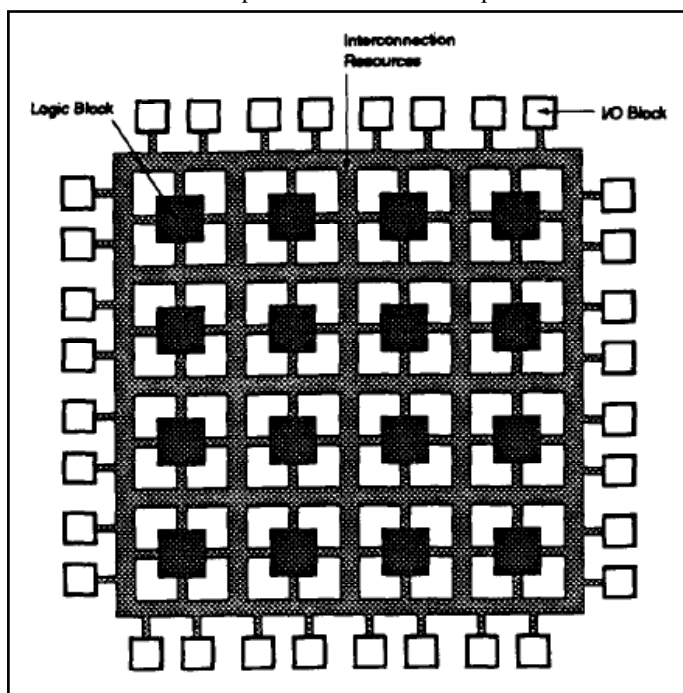


Fig. 3: FPGA Architecture

III. Related Works

In the literature, there have been several proposals devoted to the hardware implementation of FDWT/IDWT. Some proposals (M.A.Trenas et al., 2002) (et al, 2002) (Lee & Lim, 2006) (Ravasi, 2002) (P.Jamkhandi et al., 2000) (Tseng et al., 2003) addressed

the importance of flexibility and proposed programmable DWT architectures based on two types: VLSI or FPGA architecture. The VLSI architectures have large limitations in terms of flexibility and scalability compared to the FPGA architectures. Even though some recent solutions proposed programmable and scalable for either variable wavelet filters (Olkkonen & T.Olkkonen, 2010) (Lee & Lim, 2006) or the structure of FDWT, they remind, in addition to their cost, dedicated to specific algorithms and cannot be adapted to future solutions. In another hand, the existing FPGA architectural solutions are mainly ASIC like architectures and use external off-the-shelf memory components which represent a bottleneck for data access. The possibility of parallelizing the processing elements offered by FPGAs associated to a sequential access to data and bandwidth limitations do not enhance the overall computing throughput. The very powerful commercial VLIW digital signal processor obtains its performance thanks to a double data-path with a set of arithmetic and logic operators with a possibility of parallel executions and a wide execution pipeline. However, these performances are due to a high frequency working clock. Even though these DSP has a parallel but limited access to a set of instructions, the data memory access remains sequential. The performance requirement is paid by high circuit complex and power consumption. Most of work focuses on the reuse of devices like FPGAs for different applications or different partitions of one application.

In order to square up these needs, we propose a novel DWT architecture and implementation method. The proposed architecture can support multi-standard by reconfiguring the interconnection between data memories and processing elements. Moreover, the number of processing element and its working frequency could be reconfigured dynamically. A controller plays a key role as a reconfigurable interface allowing multiple accesses to local memory, external memory through a DMA and feeding the processing element in an optimal fashion. An implementation method is developed to identify parallelism level of processing element and working frequency as well as to find out the tradeoff between power consumption and performance. In comparing with others VLSI and ASIC architecture, double size of memory can be economic in using our novel architecture.

IV. Basic Concepts of DWT

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 nonstationary signals with good time localization information. 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. 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 up sampled and passed through another set of low pass and high pass filters, which is expressed as follows

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

Where $g_0(n)$ and $g_1(n)$ are respectively the low-pass and high-pass synthesis filters corresponding to the mother wavelet, and 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 $(j+1)$ Thlevel 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. The filters coefficients corresponding to this wavelet type are shown in Table 1, and have been taken from H0 and H1 are the input decomposition filters and G0 and G1 are the output reconstruction filters.

Table 1: Daubechies 8-Tap Wavelet Coefficients

H0	H1	G0	G1
-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	-0.1870
-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

V. 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 tablestructure of distributed arithmetic computation.

A. Distributed Arithmetic

Mathematical derivation of distributed arithmetic is extremely simple; a mix of Boolean and ordinary algebra. 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: equation (4)

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

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. 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.

B. Virtex FPGAs

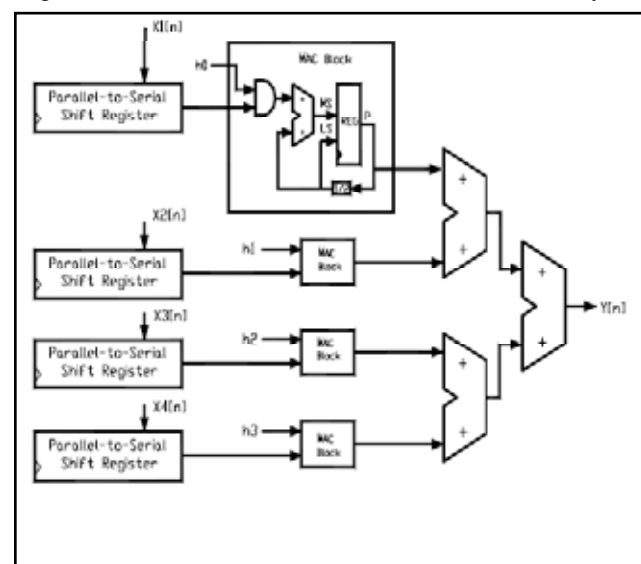
One of most advanced FPGA families in industry is the FPGA series produced by Xilinx. The Virtex user-programmable gate array comprises two major configurable elements: configurable logic blocks (CLBs) and input/output blocks (IOBs). CLBs provide the One of most provide the functional elements for constructing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

C. Internal Configuration

The basic Virtex logic element in a CLB is the slice. Two slices are present in each CLB as shown in fig. 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 these 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.

D. 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 16×1 ROM or RAM or the two LUTs may be



(a)

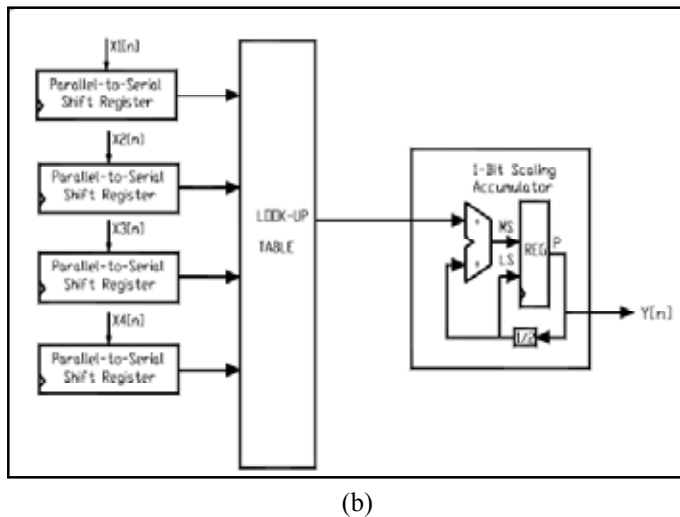


Fig. 4: Multiply Accumulate Operation (a) Conventional Implementation and (b) Distributed Arithmetic Implementation

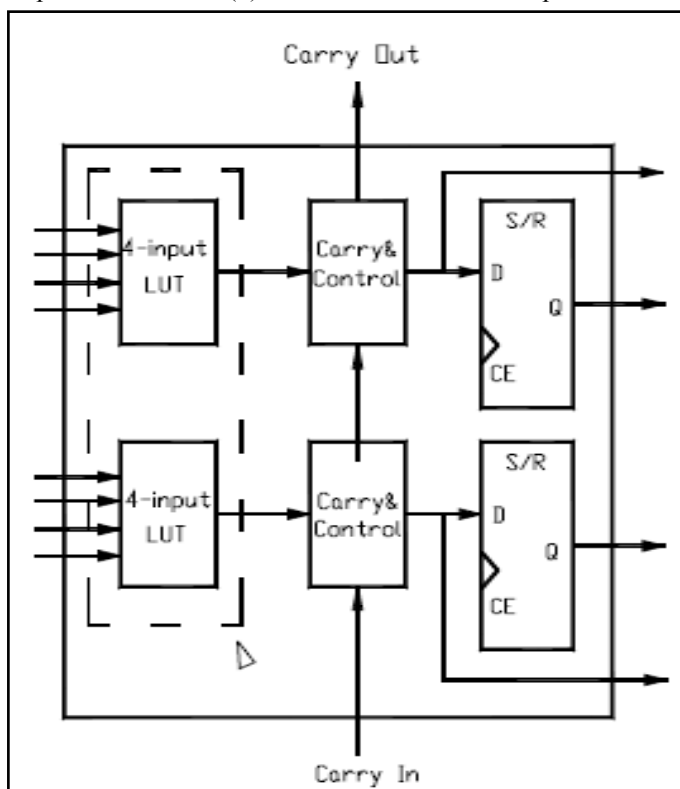


Fig. 5: Simplified Virtex Configurable Slice

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 generation 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.

E. 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 fig. 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.

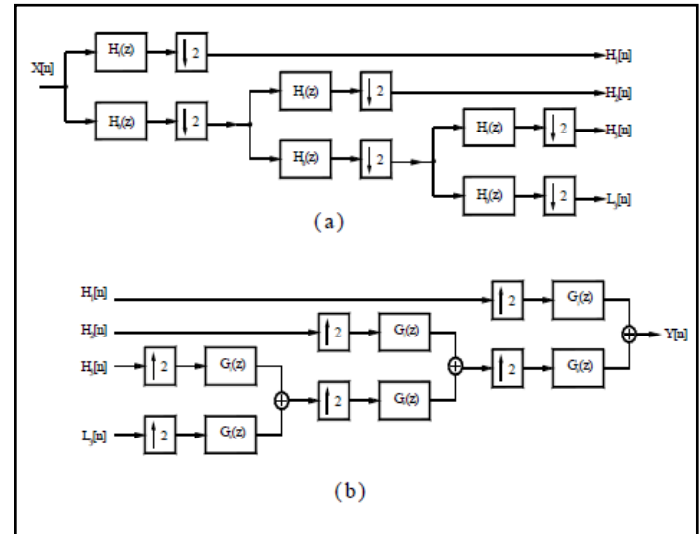


Fig. 6: Mallat's Quadratic Mirror Filter Tree (a) DWT and (b) Inverse DWT tree

F. Distributed Arithmetic FIR Filter

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

$$H(z) = \sum_{k=0}^{M-1} h[k]Z^{-k} \quad (5)$$

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. 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.

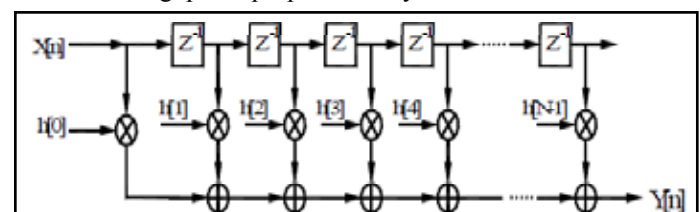


Fig. 7: Direct FIR Filter Structure

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 fig. 5.

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.

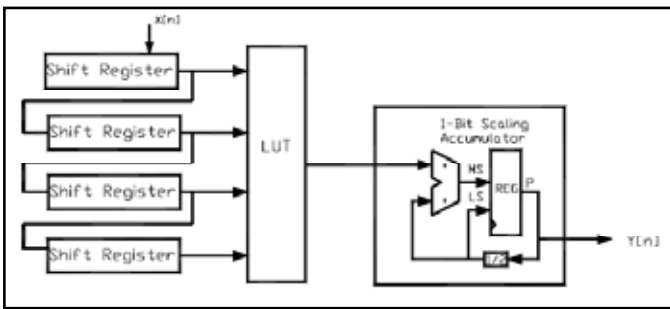


Fig. 8: Distributed Arithmetic FIR filter

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 fig. 5 into two 4-bit LUTs, and added their outputs using a two-input accumulator. The modified partitioned-LUT architecture is shown in fig. 6.

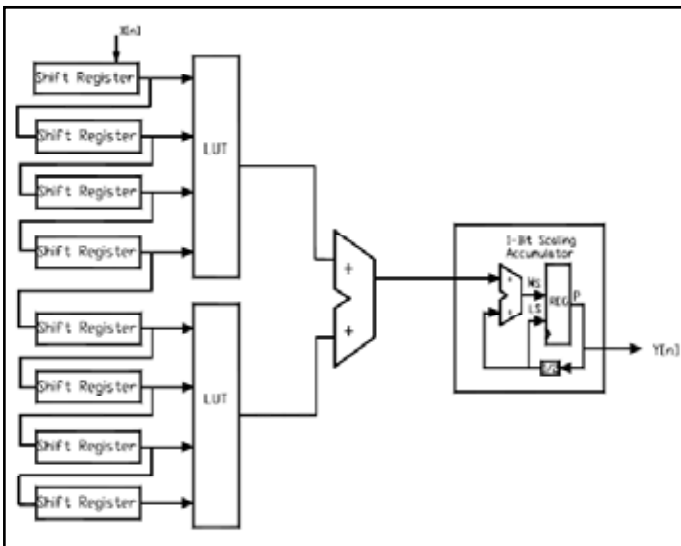


Fig. 9: Efficient Distributed Arithmetic Implementation of the FIR Filter

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 filter coefficients.

G. 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. 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 fig. 7(a). 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.

H. Inverse DWT Implementation

The basic building block of the inverse discrete wavelet transform filter bank is the interpolator which consists of an FIR filter preceded by an up-sampling operator. 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 (FFFFH) 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.

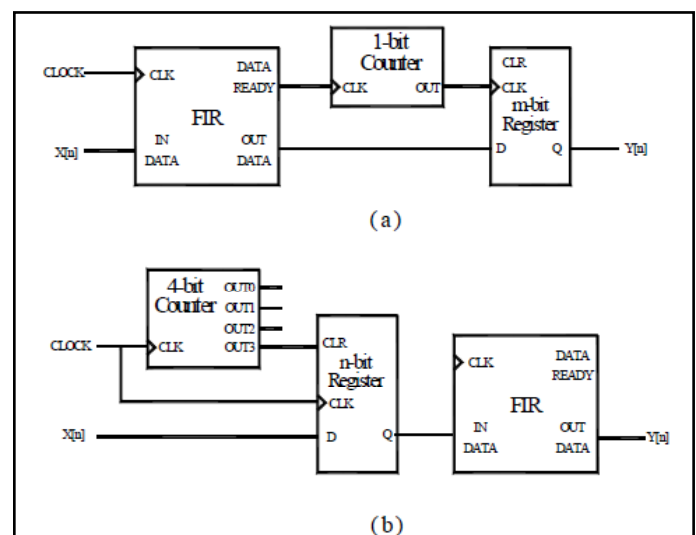


Fig. 10: Implementation of the Basic Blocks of the (a) Decimator Block and (b) Interpolator Block

I. 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 [10]. 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.

J. Performance Comparison

We implemented the discrete wavelet transform architecture shown in Figure 6 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 11, 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 amount of real-time computation. As for Virtex slice utilization, distributed arithmetic, uses less hardware resources than the conventional arithmetic, as illustrated in fig. 12.

VI. Conclusion

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 field programmable gate array devices (FPGAs), and two software implementations; one is 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 Virtex 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.

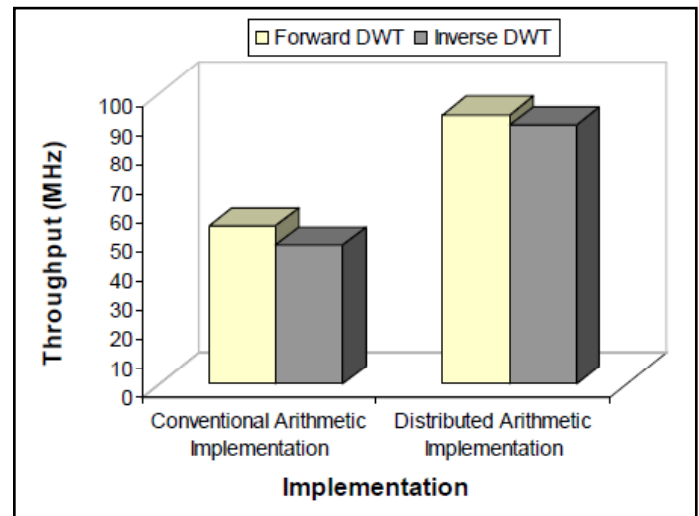


Fig. 11: Comparison Between the Throughput of Two DWT Implementations

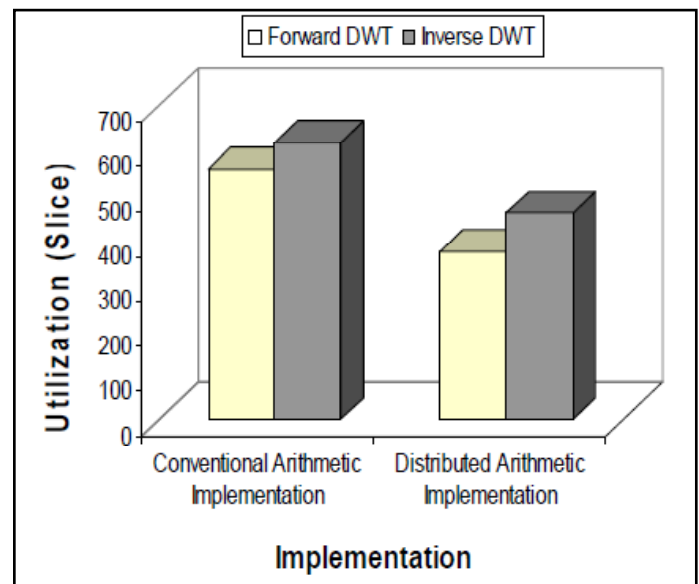


Fig. 12: Comparison Between the Utilization of Two DWT Implementations

References

- [1] Burrus, C., Gopinath, R., Guo, H., "Introduction to Wavelets and Wavelet Transforms: A Primer", New Jersey: Prentice Hall.
- [2] Daubechies, I., "Orthonormal bases of compactly supported wavelets", Communication on Pure and Applied Mathematics, 41, pp. 906-966, 1998.
- [3] Salomon, D., "Data Compression: The Complete Reference", Springer, 1999.
- [4] Mintzer, L., "The Role of Distributed Arithmetic in FPGAs", Xilinx Corporation, 1996.
- [5] Xilinx Corporation., "Xilinx breaks one million-gate barrier with delivery of new Virtex series", 1998.
- [6] Xilinx Corporation, "Virtex DataSheet", 2000.
- [7] Knowles, G., "VLSI architecture for the discrete wavelet transform", Electron Letters, 26, 15, pp. 1184-1185, 1990.
- [8] Parhi, K., Nishitani, T., "VLSI architectures for discrete wavelet transform", 1993.

- [9] IEEE Transactions on VLSI Systems, pp. 191-202.
- [10] Chakabarti, C., Vishwanath, M. 1995.
- [11] "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, pp. 759-771.
- [12] Xess Corporation. (2002), [Online] Available: <http://www.xess.com>



Alisha was born in Vijayawada, Krishna district, Andhra Pradesh, India. She is currently pursuing her Bachelor's Degree in Electronics and Communication Engineering from KL University. I am working as an intern in INCAP LIMITED from past 5 months. Her interest is in VLSI stream and EMBEDDED systems. She was a member in IETE.



BADUGU SURESH was born in Ongole. He did his B.Tech in ECE from Prakasam Engineering College, Kandukur in 2008 and M.Tech in Telematics and Signal Processing from NIT Rourkela in 2010. His interesting areas are signal processing, Image Processing and Communication systems. Presently he is working as an Assistant Professor in K.L. University, Vaddeswaram, Andhra Pradesh. He has 3 years teaching experiences and published

4 international journals in his career.