16th European Signal Processing Conference (EUSIPCO 2008), Lausanne, Switzerland, August 25-29, 2008, copyright by EURASIP

PARALLEL RECONFIGURABLE HARDWARE IMPLEMENTATIONS FOR THE LIFTING-BASED DISCRETE WAVELET TRANSFORM

Sami Khanfir, Mohamed Jemni

UTIC – Ecole Supérieure des Sciences et Techniques de Tunis
5 Ave. Taha Houssein, B.P. 56 Bab Mnara, 1008,Tunis, Tunisia
sami.khanfir@esstt.rnu.tn, mohamed.jemni@fst.rnu.tn
www.esstt.rnu.tn/utic

ABSTRACT

A novel fast scheme for Discrete Wavelet Transform (DWT) was lately introduced under the name of lifting scheme [4, 10]. This new scheme presents many advantages over the convolution-based approach [10, 11]. For instance it is very suitable for parallelization. In this paper we present two new FPGA-based parallel implementations of the DWT lifting-based scheme. The first implementation uses pipelining, parallel processing and data reuse to increase the speed up of the algorithm. In the second architecture a controller is introduced to deploy dynamically a suitable number of clones accordingly to the available hardware resources on a targeted environment. These two architectures are able of processing large size incoming images or multi-framed images in real-time. The simulations driven on a Xilinx Virtex-5 FPGA environment has proven the practical efficiency of our contribution. In fact, the first architecture has given an operating frequency of 289 MHz, and the second architecture demonstrated the controller's capabilities of determining the true available resources needed for a successful deployment of independent clones, over a targeted FPGA environment and processing the task in parallel.

Keywords – *parallel, reconfigurable, DWT, lifting, FPGA.*

1. INTRODUCTION

In the literature, first implementations of the wavelet transform were based on filters' convolution algorithms [6, 7]. This approach requires huge amounts of computational resources. In fact at each resolution, the algorithm requires the convolution of the used filters with the approximation image. A relatively recent approach uses the lifting scheme for the implementation of the DWT. This method still constitutes an active area of research in mathematics and signal processing. The lifting-based DWT scheme presents many advantages over the convolution-based approach such as the computational efficiency, the saving of memory, the integer-to-integer transform suitable for lossless image compression, the no need for boundary extension, the possibility of parallelizing the algorithm, etc.

In this context, this paper introduces two new parallel approaches for the lifting-based wavelet transform implemented using FPGA technology. Several accelerating techniques are used to achieve our goals such as the use of pipelining techniques and data reusability. The first approach proposes an architecture composed of two units for the prediction and the update of the wavelet coefficients. The two units communicate via through FIFO queues. The second approach proposes a dynamically configurable parallel architecture capable of deploying, dynamically, clones of the first architecture unit on a given FPGA environment. A controller is implemented to determine the necessary available resources allowing the successful deployment of these replicas. The simulation of these two architectures over a Xilinx Virtex-5 FPGA environment has given a maximum operating frequency of 289 MHz, for the first architecture. For the second architecture, the controller has made a successful demonstration of its capabilities of determining the true available resources on a given FPGA environment. The use of these two architectures can be extremely helpful for real-time image processing systems over large size or multi-framed images.

The outline of this paper is going to be as follows: in section 2 the theoretical basis of the convolution-based and lifting-based discrete wavelet transforms are briefly presented. The description of the lifting-based algorithm of the DWT is presented in section 3. In section 4 we present, in details, our proposed approach for the hardware implementation of the DWT lifting-based algorithm. The hardware resources utilization and the performance evaluation of the two architectures are presented in section 5. A conclusion for this paper is drawn in section 6.

2. LIFTING-BASED WAVELET TRANSFORM

Recently, a new mathematical formulation for wavelet transformation has been proposed by Swelden [4, 9, 10]. This new approach, called lifting-based wavelet transform, was primarily developed as a method to improve wavelet transform. It was extended afterward to a generic method to create so-called second-generation wavelets. The main feature of the lifting-based discrete wavelet transform scheme is to break up the high-pass and low-pass wavelet's filters into a sequence of smaller filters that in turn can be converted into a sequence of upper and lower triangular matrices [3]. The basic idea behind the lifting scheme is to use the data correlation to remove the redundancy. The lifting algorithm can be computed in three main phases, namely: the Split phase, the Predict phase and the Update phase.

2.1 Split phase

Assuming that we have a signal under the form of $\lambda_{j+1,k}$, where j and k indicate the signal decomposition level and the data element respectively; at the input, the signal is considered at the original decomposition level $\lambda_{0,k}$. In the split phase, the data set $\lambda_{0,k}$ is split into two subsets to separate the even samples from the odd ones:

$$\lambda_{-1,k} = \lambda_{0,2k} \; ; \qquad \gamma_{-1,k} = \lambda_{0,2k+1} \tag{1}$$

Conventionally, we have used the negative indices indicating that the smaller the data set is, the smaller the index will be [11]. This decomposition in even and odd samples may also be referred as the lazy wavelet transform since this procedure does not decorrelate the processed data.

2.2 Prediction phase

At this point, we will use the even subset $\lambda_{-1,k}$ to predict the odd subset $\gamma_{-1,k}$ using a prediction function $P(\lambda_{-1,k})$. The more the original data are correlated, the more the values produced by this prediction function will be close to the original $\gamma_{-1,k}$. At this point,

the difference between the predicted value of the subset and the original value is processed and replaces this latter:

$$\gamma_{-1,k} = \lambda_{0,2k+1} - P(\lambda_{-1,k}) \quad (2)$$

This procedure is known as the prediction phase or the dual lifting phase in the lifting architecture. Two types of prediction functions can be considered at this point: the piecewise linear prediction and the interpolating prediction that has an order of interpolating subdivisions denoted by N. This function is referred as the dual wavelet and N is referred as the number of dual vanishing moments [5].

2.3 Update phase

The third stage of the lifting scheme introduces the update phase. In this stage the coefficient $\lambda_{-1,k}$ is lifted with the help of the neighboring wavelet coefficients. This phase is referred as the primal lifting phase or update phase:

$$\lambda_{-1,k} = \lambda_{-1,k} + U(\gamma_{-1,k}) \quad (3)$$

Where U is the new update operator. The order of this function is the real vanishing moment $\widetilde{N}$ of the wavelet transform.

2.4 Inverse lifting transform

The inverse DWT using lifting can be derived by traversing the above steps in the reverse direction with switching the sign between additions and subtractions, applying the dual and primal lifting steps and finally applying the inverse lazy transform by upscaling the output before merging them into a single reconstructed stream.

3. DWT LIFTING-BASED ALGORITHM

For clarity purpose, we will illustrate the DWT lifting-based algorithm assuming the use of a set of data with $L = 8$ components and a filter with $N = 2$ dual vanishing moments and $\widetilde{N} = 2$ real vanishing moments. Remark that our design approach is scalable and can be implemented for arbitrary signal lengths and different number of filter coefficients.

3.1 Prediction phase

To calculate the prediction coefficients $\gamma_{j-1,k}$, the following relation has to be implemented:

$$\forall\, k \in \left[0..\frac{L}{2}-1\right], \gamma_{j-1,k} = \lambda_{j,2k+1} - \sum_{i=0}^{N-1} \lambda_{j,2(k+i)} * \alpha_{k,i} \quad (4)$$

With $\alpha_{i,k}$ are the prediction filter coefficients. This implementation is illustrated through the Figure 1.

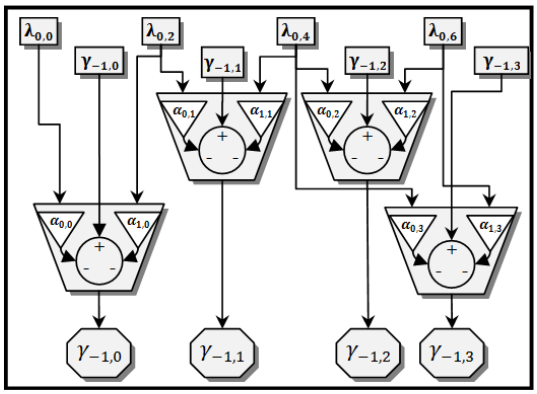

Figure 1. Prediction phase implementation

As illustrated above, the sequential version of this algorithm, consumes an important amount of computing resources and processing cycles, especially when increasing the vanishing moments of the prediction filter. In the next section, we will introduce parallel processing and data reusability to fasten the hardware implementation of this phase.

3.2 Update phase

During the update process, each previously calculated γ will update the λs. This procedure can be illustrated with the following relation:

$$\lambda_{j-1,n} = \lambda_{j-1,n} + \gamma_{j-1,i} * \beta_i \quad (5)$$

β_i are the update filter coefficients. This implementation is illustrated through Figure 2.

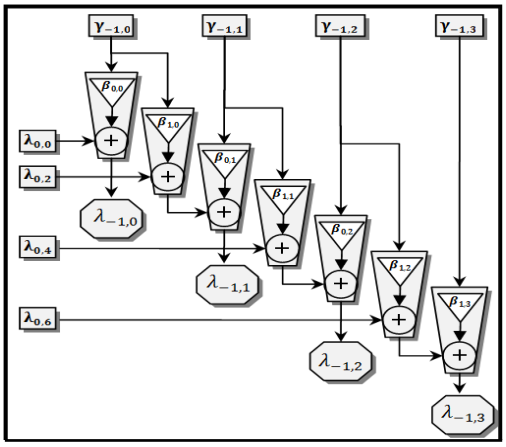

Figure 2. Update phase implementation

As illustrated above, the update's phase algorithm, consumes also an important amount of computing resources as well as processing cycles, especially when increasing the real vanishing moments of the update filters. The hardware implementation and optimization of this phase will be presented in the next section.

4. HARDWARE ARCHITECTURE FOR DWT LIFTING-BASED ALGORITHM

The goal of this work is to propose a high memory throughput architecture to treat large size images as well as real-time DWT processing for video treatment. In the following four sub-sections, we will describe our parallel approach for every unit followed by the dynamically reconfigurable parallel hardware architecture we proposed to implement our approach.

4.1 The Prediction Unit

Our implementation of the prediction phase of the DWT lifting-base algorithm is based on a pipelined architecture as illustrated in the Figure 3. For clarity purpose, we will use the same example as in section 3 ($L = 8$ components and $N = 2$ dual vanishing moments). Our approach exploits the fact that in the processing of two consecutive values of λ, use some of the coefficients λ are commonly reused during the calculation (Figure 1), thus for the processing of the next γ coefficient, only one new λ coefficient is read from the memory, the preceding λ coefficients, involved during the previous calculations, have to be temporarily stored in the buffer for reuse. In this context, the use of the pipelining technique would be with great support for this problem. Effectively, we have implemented a pipeline of N stages for the λ inputs coefficients. $N - 1$ cycles would be involved to fill-in the pipeline during the initialization process followed by the parallel processing of $N * \lambda$ coefficients driven to the multiplier. To ensure the accessing of N filter coefficients concurrently, we used separate banks of RAM to store the filter coefficients. To ensure the parallel processing of the unit, we have to process the reading of both λ and γ input coefficients at the same time. This means that we have to access different locations of the storage memory of the image at the same time. We have used for this purpose true dual port memories with separate undependably addressable input/output ports configured directly in

the Xilinx FPGA processing core. After processing the predicted γ coefficient, we have to write it back at the same initial memory location, rather than at another memory location, to ensure efficiency and speed up of this architecture. In fact, the reading of the inputs data, λ and γ, from the memory, has to be done at the same time as the writing of the outputs into the memory. This is quite difficult since both ports of the dual ports of the RAM are already involved in the reading process. To counterpart this difficultly, we imposed to the RAM to operate at twice the frequency of the entire design. Finally, when considering the treatments over the boundaries, the processing unit has to stop when attending the signal boundaries and consider only the corresponding filter coefficients and the associated coefficient of γ. For this purpose we have added some enabling signals into the pipeline process.

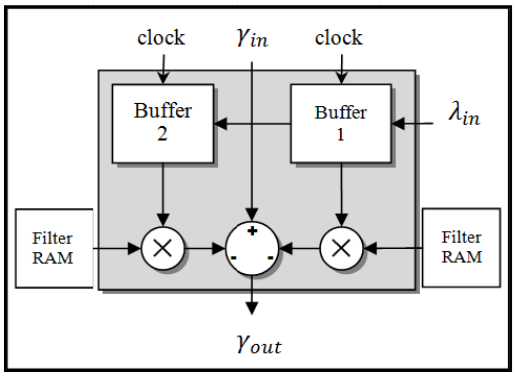

Figure 3. Parallel prediction unit architecture

4.2 The update unit

As done for the prediction phase, our parallel approach for the update unit is based on a pipelined architecture as described in the Figure 4. We will use also the same example given in section 3 ($L = 8$ components and $\tilde{N} = 2$ real vanishing moments).

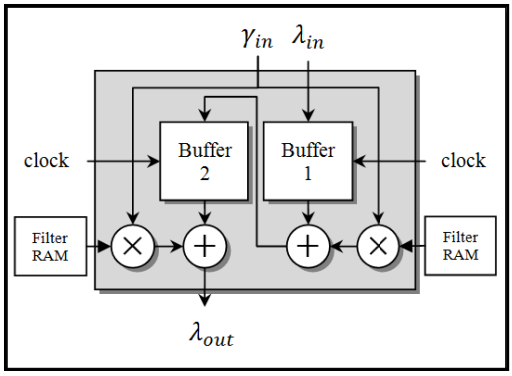

Figure 4. Parallel update unit architecture

In this phase the treatment starts by an initialization step to feel the pipeline register with the initial data. Afterward, the content of the λ registers has to be shift to the left-hand side; at the same time the acquisition of the data, from the RAM, has to continue. The filter coefficients, corresponding to the last λ with the first γ, have to be loaded, via the update coefficient, with the λ_{in} inputs at the same time as the filling of the last λ. At the adder's output, after being processed, the updated λ coefficients are ready to be stored. While performing all these operations, to consider the exceptions of the boundary treatments, we have used a special configuration for processing the γ coefficients in questions. We have used a reset signal to stop the pipeline and freeze the λ coefficients from being shifting. The output sample is calculated and issued via λ_{out}. When the λ coefficients are available at the output, they are written back in the memory.

4.3 Inverse prediction and update implementation

To perform the inverse prediction and update phases, we can notice that just some small changes have to be applied to both units to obtain the desired reverse result. In fact we have just to substitute the addition process with a subtraction process in the prediction unit and vice versa in the update unit. Therefore the same prediction and update units will be used for both forward and inverse transform by selectively alternating a control signal to set the scheme to forward or inverse processing. This signal applies a control after the multiple-inputs adder for performing either addition or subscription in both units.

4.4 Unified unit for DWT lifting-based prediction and update processing

To conceive a unified unit for both prediction and update DWT lifting based processing phases, we have used a FIFO (First-In First-Out) buffer to synchronize the communication between the two units. In fact a simple parallel implementation of both prediction and update units would overload the memory bandwidth. Indeed, the parallel execution of both units implies six memory accesses per cycle (three accesses for the prediction units and three others for the update unit). We used a FIFO buffer, between the two units, in order to have only four concurrent accesses to memory (two accesses for the prediction module, one access for the outputs and one access for the update). The Figure 5 illustrates this unified unit based on a FIFO buffer use.

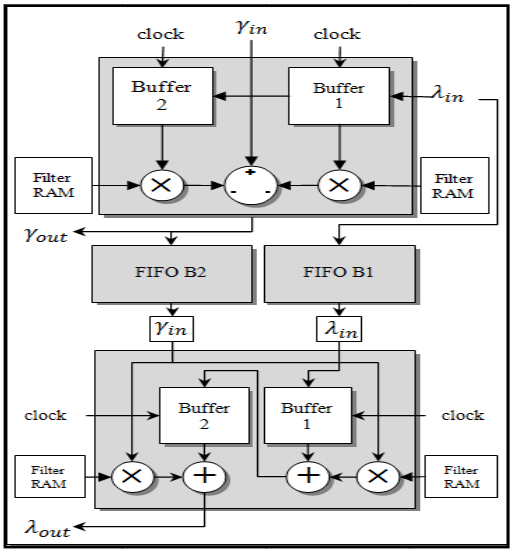

Figure 5. Parallel unified unit architecture

We can easily notice that the input of the update unit uses the same input λ coefficient, of the prediction unit, at a different time rate. It is indeed obvious that we cannot connect the RAM to both inputs of the prediction unit and update unit. The insertion of a FIFO buffer B1, before the λ input of the update module, allows this latter to reuse the λs that has been involved in the production phase. The FIFO buffer B2 absorbs the unequal delivery and compensation rates of data at the beginning and at the end of the prediction and update phases. When considering the inverse transform, the synchronization scheme implies to reverse the above described synchronization process by providing data to the inputs of the prediction unit from the FIFOs and receiving data from the RAM for the update unit.

4.5 Dynamic parallel hardware architecture for lifting-based DWT algorithm

To increase the performance of our implementation, we have used the unified unit, described above, in a dynamic and parallel architecture. This latter is capable of treating several tiles of the image in parallel. Our approach is based on a dynamic reconfiguration in order to use the available resources at the deployment step. In other words, our system verifies the amount of the available resources present in the hardware in question, before any deployment, and then clones the unified unit, described above, following the connection architecture depicted in Figure 6.

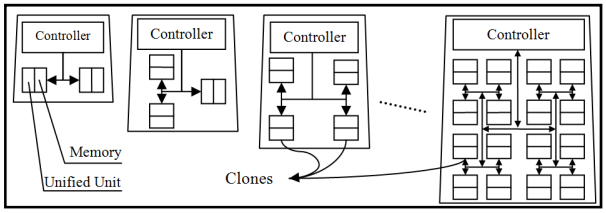

Figure 6. Dynamic parallel hardware architecture of the DWT lifting-based algorithm

Depending on the acquired parameters, from the hardware, where the architecture is going to be deployed, the system calculates the maximum number of clones, of the DWT lifting-based unified unit, and builds the adequate connection architecture. For this purpose, we have conceived a global controller to insure the synchronization and the communication (if needed) between the different units. Afterward, the controller builds the connection architecture following the parameters that it acquired from the hardware. The final step of the deployment is the building of the different clones at the tail of each created connection as shown in Figure 6. Each clone will have its own memory, based on the cascading asynchronous dual-port block RAM: For our implementation we have used adjacent combined block RAMs memory. The Figure 6 illustrates a fully deployed cloned architecture based on the unified DWT lifting-based unit. This same figure, also illustrates the hierarchical connection architecture between the different units, their associated memory and the controller. After the successful deployment, each clone will work independently from the others. In fact the controller will assign different tiles to each clone (their size is dynamically fixed by the controller depending on the size of the initial image and the number of deployed clones). Due to the diversity of content of the processed image and therefore the diversity content of each processed tile, a given clone can finish its processing before another. We have used the First Finished First Served strategy to distribute the jobs over the clones. When two clones finish their job at exactly the same time, the first served would be the nearest one to the controller.

5. EXPERIMENTAL RESULTS

5.1 Hardware resources utilization

We have implemented the above described architectures using VHDL description language and schematic-based design. The synthesis of these architectures was performed using ISE foundation design tool (version 9.1i). We have used a Xilinx ML501 evaluation platform based on the Xilinx Virtex-5 FPGA, XC5VLX50T-1FFG676 core to implement our architectures [13, 14]. The Figure 7 illustrates the hardware resources utilization considering the use of an image with 128x64 pixels size and an eight bits gray-scale. We have used the (9/7) wavelet filter used in the JPEG2000 standard.

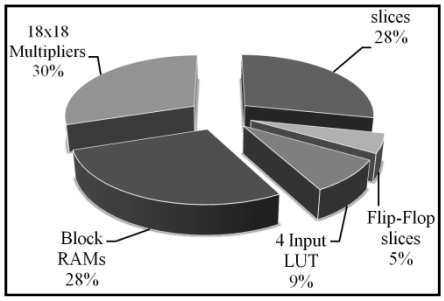

Figure 7. Hardware resources utilization

This implementation has given a maximum operating frequency of 289MHz for one single unified unit. In fact, from the experimental simulations, we could remark that the implementation of one single unit consumed only 849 Slices of the 7200 available ones, 607 Flip-Flop slices and 1046 4-Input LUTs from the 28800 available ones, 14 BRAMs from the 120 available ones and 6 18x18 Multipliers from the 48 available ones. All the statistics, shown in Figure 7, are exactly preserved as they are for larger images processing except for the Bank RAMs that are dynamically modified accordingly to the image and filter size.

5.2 Performance evaluation

To evaluate the performance of our architectures we have considered two scenarios for the single unit based implementation and three others for the parallel dynamic one. Our evaluation is based on the following criteria: the number of cycles per pixel, the number of images per second in the transform time. We measure the time to perform the discrete wavelet transform on an entire image including all the required data transfers. We have compared all the collected results to recently related works such as [1, 2, 8, 12]...

5.2.1 Single-unit based implementation evaluation

We have used two scenarios of evaluation:

- Diversifying the degree of the polynomial filters, and fixing the image size. Figure 8 presents the performance results obtained with different polynomial degrees of filtering and an image of 1024x768 pixels size.

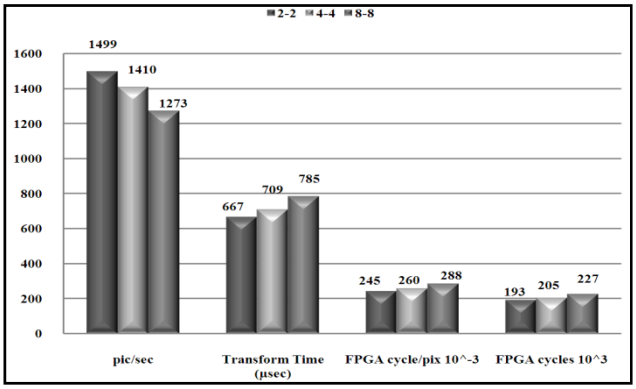

Figure 8. Single-unit based implementation evaluation for different degrees of polynomial filters

We can notice that the more the polynomial degree of the used filters increases, the more the transform time increases and the more the number of treated images per second decreases.

Diversifying the image sizes and maintaining fixed the polynomial degree of the used filter. Figure 9 presents the performance results using a 2-2 polynomial filter. We can notice that the more the image size increases, the more the FPGA cycles/pixel decreases.

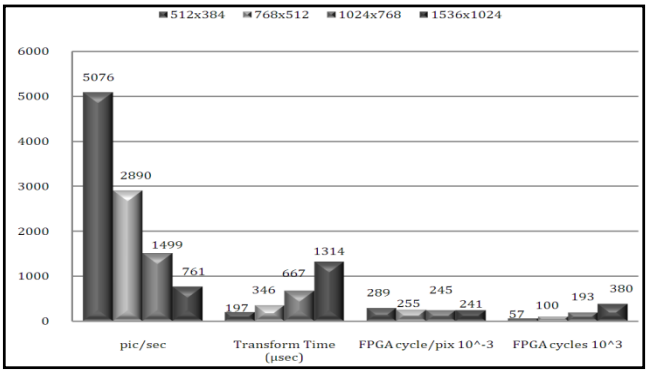

Figure 9. Single-unit based implementation evaluation for different image's sizes

Table 1 illustrates a comparison to other related works such as [1, 2, 8, 12]. These results demonstrate the efficiency of our implementation from hardware resources and maximum operating frequency point of view.

Ref.	FPGA core	Decomp. Levels	Slices	Freq (MHz)
[8]	APEX20K E	2	7726	66.8
[1]	XCV300	2	853	89.1
[12]	XCVE2000	2	1402	159.51
[2]	XCV2P20	2	1907	201.09
Our work	XC5VLX50T	2	849	289

Table 1. Performance comparison with existing FPGA implementations

5.2.2 Dynamic parallel hardware architecture performance evaluation

We have considered three cases for the performance evaluation:

- First case: the hardware has completely all its resources fully available for the implementation.
- Second case: the hardware has an already running application consuming 37% of the available hardware resources: to demonstrate the controller's capability of adapting the deployment in function of the available material resources.
- Third case: the hardware has one running job consuming only 24% of the resources. This job consumes a lot of memory banks allowing the deployment of just one unique clone: in order to check the controller's capability of recognizing that the available memory resources are not sufficient for more than one clone deployment even if there are remaining free resources.

Table 2 illustrates the different results concerning this implementation. These results show that the controller is able to determine the necessary number of clones that can be possibly deployed into a targeted architecture.

	Case 1	Case 2	Case 3
Available resources	100%	63%	76%
slices	7200	4536	5472
Flip-Flop slices	28800	18144	21888
4 Input LUT	28800	18144	21888
Block RAMs	120	75	22
18x18 Multipliers	48	30	36
Number of clones	8	5	1

Table 2. Clones' deployment statistics for different cases of resources availabilities

6. CONCLUSION

We have introduced in this paper a novel hardware implementation of the discrete wavelet transform based on the lifting scheme. We have used, for the purpose of speeding up the performances, several accelerating techniques such as pipelining, parallel module operation and data reuse to implement a unified unit. This latter is composed of one prediction based processing unit and one update based processing unit connected through FIFO blocks. We have also conceived a dynamically reconfigurable parallel hardware architecture capable of dynamically deploying clones of the unified unit on an FPGA environment by determining the necessary available resources allowing the successful deployment of these clones. The performance evaluation has proven the efficiency of our approach. In fact the simulation of a single processing unit on a Xilinx Virtex-5 FPGA environment has given an operating frequency of 289MHz. The implementation of the parallel reconfigurable version of the DWT lifting-based processing unit demonstrated the controller's capabilities of determining the true available resources needed for a successful deployment over a given FPGA environment. Finally the use of these two architectures could be extremely helpful for real-time image processing systems of large size images.

7. REFERENCES

[1] A. Al-Haj, "Fast Discrete Wavelet Transformation Using FPGAs and Distributed Arithmetic," International Journal of Applied Science and Engineering, 2003, 160-171

[2] S. L. Bishop, S. Rai, B. Gunturk, J. L. Trahan, R. Vaidyanathan, "Reconfigurable Implementation of Wavelet Integer Lifting Transforms for Image Compression", Reconfigurable Computing and FPGAapos, 2006. IEEE International Conference on ReConFig, pp1 – 9, Sept. 2006.

[3] R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo, "Losless image compression using integer to integer wavelet transforms," International Conference on Image Processing (ICIP), Vol. I, pp. 596–599, IEEE Press, 1997

[4] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting schemes," J. Fourier Anal. Appl., vol. 4, no. 3, pp. 247–269, May 1998.

[5] G. Fernandez, S. Periaswamy and W. Sweldens, "LIFTPACK: A software package for wavelet transforms using lifting," in Wavelet Applications in Signal and Image Processing IV, pp. 396–408, Proc. SPIE 2825, 1996

[6] A. Grapes, "An introduction to wavelets," IEEE Computational Science and Engineering, Summer 1995, vol. 2, num. 2

[7] C. Herley and M. Vetterli, "Orthogonal time-varying filter banks and wavelets," in Proc. IEEE Int. Symp. Circuits Systems, vol. 1, May 1993, pp. 391-394

[8] S. Masud and J.V. McCanny, "Rapid design of biorthogonal wavelet transforms", IEE Proceedings of Circuits, Devices and Systems, Volume: 147 Issue: 5, 2000, pp. 293 – 296

[9] W. Sweldens and P. Schröder. "Building your own wavelets at home," In Wavelets in Computer Graphics , pages 15- 87. ACM SIGGRAPH Course notes, 1996

[10] W. Sweldens, "The lifting scheme: a custom-design construction of biorthogonal wavelets," Applied and Computaional Harmonic Analysis, vol. 3, no. 15, pp. 186–200, 1996.

[11] W. Sweldens, "The lifting scheme: A construction of second generation wavelets," SIAM Journal on Mathematical Analysis, Volume 29, Number 2, pp. 511-546, 1998

[12] I. Uzun and A Amira, "Design and FPGA implementation of high-speed discrete biorthogonal wavelet transforms," 13th European signal processing conference EUSIPCO 2005, Sep. 2005

[13] Virtex-5 Family Overview, DS100 (v3.4), December 18, 2007

[14] Xilinx ML501 Evaluation Platform User Guide, UG226 (v1.2) November 26, 2007