Reconfigurable Hardware Implementations for Lifting-Based DWT Image Processing Algorithms

Sami KHANFIR, Mohamed JEMNI
Research Unit of Technology of Information and Communication
Ecole Supérieure des Sciences et Techniques de Tunis, Tunis, Tunisia
sami.khanfir@esstt.rnu.tn, mohamed.jemni@fst.rnu.tn

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.

1 Introduction

This Multiresolution theory incorporates and unifies techniques from a variety of disciplines, including subband coding [9] and quadrature mirror from signal processing [6], and pyramidal image processing [9]. The multiresolutional analysis analyzes the signal at different frequencies with different resolutions. As an alternative to the Fourier transform, the wavelet transform was developed to overcome the resolution difficulties.

The Fourier transform has been, for many years, the mainstay of transform based image processing [9]. However, with the wavelet transform, it became easier to compress, transmit, and analyze images. In the one dimensional wavelet transform, an assembly of wavelet is generated based on one unique function called the wavelet mother.

As an alternative to the Discrete Cosine Transform (DCT), the technology of wavelet-base compression offers a larger smoothness to the level of signal analysis and allows better adaptation to the local properties of the image. This technology constitutes today a promising way of research. Many standards like JPEG2000 and SPIHT are based on the wavelet transform. Furthermore, the software implementations of the discrete wavelet transform, although greatly flexible, appear to be performance bottlenecks in real-time systems. Hardware implementations, in contrast, offer high-performance but poor flexibility. For these reasons the use of reconfigurable hardware to implement this technology can be considered as a good solution for real-time processing systems.

In the literature, early implementations of the wavelet transform were based on filters’ convolution algorithms. This approach requires a huge amount of computational resources. In fact at each resolution, the algorithm requires the convolution of the filters used 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 computational efficiency, saving of memory, integer-to-integer transform suitable for lossless image compression, no need for boundary extension, 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 through FIFO queues. The second approach proposes a dynamically configurable parallel architecture capable of, dynamically deploying 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 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 detail, our proposed approach for the hardware implementation of the DWT lifting-based algorithm. The hardware resource 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 Wavelet Transform

The wavelet transform replaces the sinusoid-based Fourier transform with a family of translation and dilation of the same function (the wavelet mother). The translation and dilation parameter are the two arguments of the wavelet transform. If the wavelet mother is represented by \( \psi(t) \), the other wavelets \( \psi_{a,b}(t) \) can be represented as:

\[
\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}} \psi\left(\frac{t-b}{a}\right)
\]

(1)

Where \( a \) and \( b \) are two arbitrary real numbers representing the dilation and translation, respectively, in the time axis [9].

2.1 Lifting-based wavelet transform

Recently, a new mathematical formulation for wavelet transformation has been proposed by Swelden [4, 10]. This new approach, called lifting-based wavelet transform, was primarily developed as a method to improve the wavelet transform. It was extended afterward to a generic method to create a 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 filters into a sequence of smaller filters that in turn can be converted into a sequence of upper and lower triangular matrices. The basic idea behind the lifting scheme is to use data correlation to remove the redundancy. Some of the advantages of this reformulation of the DWT includes "in-place" computation of the DWT, integer-to-integer wavelet transform (IWT), symmetric forward and inverse transform, no signal boundary extension requirements, etc.

Given a complementary filter pair \((\hat{h}, \hat{g})\), there always exist Laurent polynomials \( \hat{s}_i(z) \) and \( \hat{\ell}_i(z) \) for \( i \leq i \leq n \) and a polyphase matrix \( \hat{P}(z) \) that can be factorized into a finite sequence of alternating upper and lower triangular matrices as follows:

\[
\hat{P}(z) = \left\{ \prod_{i=1}^{m} \begin{bmatrix} [1] & \hat{s}_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \hat{\ell}_i(z) & 1 \end{bmatrix} \right\} \begin{bmatrix} K & 0 \\ 0 & 1 \end{bmatrix}
\]

(2)

Where \( K \) is a constant that acts as a scaling factor. Hence the lifting-based forward wavelet transform essentially first applies the lazy wavelet transform on the input stream (split into even and odd samples), then alternately executes primal and dual lifting steps and finally scales the two output streams by \( \frac{1}{K} \) and \( K \) respectively to produce low-pass and high-pass subbands. Computing the upper triangular matrix is known as primal lifting and similarly, computing the lower triangular matrix is known as dual lifting [10, 11]. The lifting algorithm can be computed in three main phases, namely: the Split phase, the Predict phase and the Update phase, as illustrated in Figure 1.

---

**Figure 1.** Split, predict and update phases of the lifting based DWT
2.1.1 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 will be considered at the original decomposition level $\lambda_{0,k}$. In this split phase, the data set $\lambda_{0,k}$ is split into two subsets to separate the even samples from the odd ones:

$$\lambda_{-1,k} = \lambda_{0,2k} ; \quad y_{-1,k} = \lambda_{0,2k+1}.$$  \hspace{1cm} (3)

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.

2.1.2 Prediction phase. At this point, we will use the even subset $\lambda_{-1,k}$ to predict the odd subset $y_{-1,k}$ 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 $y_{-1,k}$. At this point, the difference between the predicted value of the subset and the original value is processed and replaces this latter:

$$y_{-1,k} = \lambda_{0,2k+1} - P(\lambda_{-1,k})$$  \hspace{1cm} (4)

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:

- Piecewise linear prediction: The odd samples are predicted for every point as the average of its two even neighbors, $y_{-1,k} = Y_{0, k+1}$.
- Interpolating prediction: This model uses the same basic idea as the piecewise linear prediction but uses 2 or more neighbors to both side and an interpolating function to predict the odd samples. The order of the interpolating subdivision, is denoted by $N$. This function is referred as the dual wavelet and $N$ is referred as the number of dual vanishing moments [5].

2.1.3 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:

$$\lambda_{-1,k} = \lambda_{-1,k} + U(y_{-1,k})$$  \hspace{1cm} (5)

Where U is the new update operator.

Note that this procedure combined to the prediction phase implies that the odd samples are calculated from the even samples and even samples are calculated from the updated odd samples. This phase processes the scaling function from the wavelet coefficients calculated in the prediction phase to maintain some properties among all the $\lambda$ coefficients throughout every level. The order of this function is the real vanishing moment $N$ of the wavelet transform.

2.1.4 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 purposes, 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 $\bar{N} = 2$ real vanishing moments. Notice 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 $y_{j-1,k}$, the following relation has to be implemented:

$$\forall \; k \in \left[0, \frac{L}{2} - 1\right], \quad y_{j-1,k} = \lambda_{j,2k+1} - \sum_{l=0}^{N-1} \lambda_{j,2(k+l)} \cdot \alpha_{kl}$$  \hspace{1cm} (6)

With $\alpha_{kl}$ are the prediction filter coefficients. This implementation is illustrated through the Figure 2.

![Figure 2. Prediction phase implementation](image)

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 increase the hardware implementation speed of this phase.
3.2 Update phase

During the update process, each previously calculated $\gamma$ will update the $\lambda$s. This procedure can be illustrated with the following relation:

$$\forall k \in [0, \frac{L}{2} - 1], \lambda_{j-1,k} = \lambda_{j,2k} + \gamma_{j-1,i} \ast \beta_{k,i}$$  (7)

$\beta_{k,i}$ are the update filter coefficients. This implementation is illustrated through Figure 3.

As illustrated, 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 4. For clarity purposes, 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 $\lambda$, $s$ some of the coefficients $\lambda$ are commonly reused during the calculation (Figure 2). Thus, for the processing of the next $\gamma$ coefficient, only one new $\lambda$ coefficient is read from the memory. The preceding $\lambda$ 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 of great help. Effectively, we have implemented a pipeline of $N$ stages for the $\lambda$ input coefficients. $N - 1$ cycles would be involved to fill-in the pipeline during the initialization process followed by the parallel processing of $N \ast \lambda$ 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 $\lambda$ and $\gamma$ 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 independently addressable input/output ports configured directly in the Xilinx FPGA processing core.

After processing the predicted $\gamma$ 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 input data, $\lambda$ and $\gamma$, from the memory, has to be done at the same time as the writing of the output 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 counteract this difficulty, we make the RAM 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 \( \gamma \). For this purpose we have added some enabling signals into the pipeline process.

### 4.2 The update unit

As was done for the prediction phase, our parallel approach for the update unit is based on a pipelined architecture as described in the Figure 5. We will use also the same example given in section 3 (\( L = 8 \) components and \( \bar{N} = 2 \) real vanishing moments).

In this phase the treatment starts by an initialization step to fill the pipeline register with the initial data. Afterward, the content of the \( \lambda \) registers has to be shifted 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 \( \lambda \) with the first \( \gamma \), have to be loaded, via the update coefficient, with the \( \lambda_{in} \) inputs at the same time as the filling of the last \( \lambda \). At the adder’s output, after being processed, the updated \( \lambda \) 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 \( \gamma \) coefficients in question. We have used a reset signal to stop the pipeline and freeze the \( \lambda \) coefficients from being shifted. The output sample is calculated and issued via \( \lambda_{out} \). When the \( \lambda \) coefficients are available at the output, they are written back in the memory.

![Figure 5. Parallel update unit architecture](image)

### 4.3 Inverse prediction and update implementation

To perform the inverse prediction and update phases, we notice that only some small changes have to be applied to both units to obtain the desired reverse result. In fact we only have 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 to perform either addition or subtraction in both units.

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

To create 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). Figure 6 illustrates this unified unit based on a FIFO buffer use.

![Figure 6. Parallel unified unit architecture](image)

We can easily notice that the input of the update unit uses the same input \( \lambda \) coefficient, like 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 \(\lambda\) input of the update module, allows this latter to reuse the \(\lambda\)s that have 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 reversing 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. The latter is capable of treating several tiles of the image in parallel. Our approach is based on 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 7. 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 connection architecture. For this purpose, we have introduced 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 7. 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 RAM memory. Figure 7 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. The use of dynamic reconfiguration ports (DRP) allows the dynamic reconfiguration of the functional blocks of this architecture depending on the needs and the available resources.

![Figure 7. Dynamic parallel hardware architecture of the DWT lifting-based algorithm](image)

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 determined dynamically 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. Figure 8 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. 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 [13, 14, 15]. All the statistics, shown in Figure 8, 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 (in clock cycles and in microseconds respectively). 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 9 presents the performance results obtained with different polynomial degrees of filtering and an image of 1024x768 pixels size.

We can notice that as the polynomial degree of the used filters increases, so does the transform time while the number of treated images per second decreases.
- Diversifying the image sizes and maintaining fixed the polynomial degree of the used filter. Figure 10 presents the performance results using a 2-2 polynomial filter. We notice that as the image size increases, the FPGA cycles/pixel decreases.

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.

### Table 1. Performance comparison with existing FPGA implementations

<table>
<thead>
<tr>
<th>Ref. to related works</th>
<th>FPGA core</th>
<th>Decomp. Levels</th>
<th>Slices</th>
<th>Freq (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[8]</td>
<td>XCV300</td>
<td>2</td>
<td>785</td>
<td>85.49</td>
</tr>
<tr>
<td>[1]</td>
<td>XC2200</td>
<td>2</td>
<td>1402</td>
<td>159.51</td>
</tr>
<tr>
<td>[2]</td>
<td>XC2200</td>
<td>2</td>
<td>1402</td>
<td>159.51</td>
</tr>
<tr>
<td>Our work</td>
<td>XCVLX5C</td>
<td>2</td>
<td>849</td>
<td>289</td>
</tr>
</tbody>
</table>

5.2.2 Dynamic parallel hardware architecture performance evaluation. We have considered three cases for the performance evaluation:
- First case: the hardware has 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 to recognize 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 implementation’s results.
Table 2. Clones’ deployment statistics for different cases of resources availabilities

<table>
<thead>
<tr>
<th>Available resources</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>slices</td>
<td>100%</td>
<td>63%</td>
<td>76%</td>
</tr>
<tr>
<td>Flip-Flop slices</td>
<td>7200</td>
<td>4336</td>
<td>5472</td>
</tr>
<tr>
<td>4 Input LUT</td>
<td>28800</td>
<td>18144</td>
<td>21888</td>
</tr>
<tr>
<td>Block RAMs</td>
<td>120</td>
<td>75</td>
<td>22</td>
</tr>
<tr>
<td>18x18 Multipliers</td>
<td>48</td>
<td>30</td>
<td>36</td>
</tr>
<tr>
<td>Number of clones</td>
<td>8</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

These results show that the controller is able to determine the necessary number of clones that can be possibly deployed into a targeted architecture.

6 Conclusion

In this paper, we have introduced a novel hardware implementation of the discrete wavelet transform based on the lifting scheme. For the purpose of speeding up the performances, we have used several accelerating techniques such as pipelining, parallel module operation and data reuse to implement a unified unit. The 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 Acknowledgment

Special thanks to Professor Panos Papamichalis, from the Southern Methodist University, Dallas, Texas, for his reading to this work.

8 References


