Reconfigurable DWT Unit Based on Lifting

Georgi Kuzmanov, Bahman Zafarifar, Prarthana Shrestha, Stamatis Vassiliadis
Computer Engineering Lab, Delft University of Technology,
P.O. Box 5031, 2600 GA Delft, The Netherlands
Phone: +31(0)15 278 7364 E-mail: G.Kuzmanov@ET.TUDelft.NL

Abstract— **At algorithmic level, the so-called lifting scheme represents the fastest implementation of the Discrete Wavelet Transform (DWT). In this paper, a hardware accelerator for the lifting scheme is described. A lifting-based DWT unit was implemented in reconfigurable hardware, namely the Xilinx VIRTEX II FPGA. The hardware module achieves the acceleration using techniques as pipelining, data re-usability, parallel operating sub-units and some specific features of the Xilinx FPGAs. A VHDL model was developed and synthesized with the implementation tools of the FPGA vendor. Synthesis results prove the feasibility of a 50 MHz FPGA implementation, allowing processing rates between 85 and 1087 pictures per second for a range of standard picture dimensions. To estimate the performance gains from the hardware module, we compare these results against a pure software implementation of the algorithm from the LIFTPACK software package. For picture size of 720 x 560 pixels, assuming clock frequency of 50 MHz for the hardware module, simulation results indicate a speed-up of over 5 times versus a pure software realization on a 1 GHz general purpose MIPS processor. Moreover, the speed-up grows for larger images and filters with higher degrees. The hardware area costs are estimated to be 985 Virtex II CLB slices, 669 Flip-Flops, 22 Block RAM and 8 multiplier blocks for a basis structure. The design is generic and scalable, which allows better performance when more parallel sub-units are implemented.**

Keywords— **Data compression, DWT, Lifting scheme, Reconfigurable unit**

I. INTRODUCTION

The Discrete Wavelet Transform (DWT) has become a basic encoding technique for recent data compression algorithms. In Wavelet Transform, dilations and translations of a mother wavelet are used to perform a spatial/frequency analysis on the input data. Varying the dilation and translation of the mother wavelet, produces a customizable time/frequency analysis of the input signal. Compared to traditional DCT-based processing, DWT yields higher compression ratios and better visual quality. For example, the DCT-based JPEG algorithm yields good results for compression ratios up to 10:1. As the compression ratio increases further, coarse quantization of the DCT coefficients causes blocking effects in the decompressed image. In contrast, for Wavelet Transform followed by Embedded Zero Tree encoding algorithm, compression ratios of the order of 100:1 have been achieved, still yielding reconstructed images at acceptable quality. Software implementations of the DWT, however, although greatly flexible, appear to be the performance bottlenecks in real-time systems. Hardware implementations, in contrast, offer high performance but poor flexibility. A compromise solution of this dilemma is the reconfigurable hardware implementation. It allows more flexibility to be preserved, accompanied by speed-up gains from the reconfigurable hardware operation.

At algorithmic level, the so-called *lifting scheme* describes an efficient implementation of the Wavelet Transform. In the lifting scheme, half of the data samples are used to predict the other half. The transform process is split into three phases, which are iterated until all samples are predicted. Applying the inverse transform in the lifting scheme is also very easy and, as long as the transform coefficients are not quantized, it will always result in a perfect reconstruction of the original picture, regardless of the precision of the applied arithmetic. Moreover, it is possible to use integer arithmetic without encountering problems due to finite precision or rounding.

This paper proposes a novel hardware design of the lifting based Wavelet Transform. The unit is implemented in reconfigurable (FPGA) technology and is meant to be integrated as an extension to a general-purpose processor in a custom-computing platform [2]. To maximize the performance of the design, we utilize different techniques such as pipelining, parallel operating modules, data reusability and specific features of the chosen FPGA platform (Xilinx Virtex II). The software package Lift pack [1] was used for performance evaluation of the design. We first optimized the software for integer arithmetic and used it as a benchmark. The benchmark was then executed using the cycle accurate simulator Sim-outorder, from the SimpleScalar toolset [3]. Finally, we compared the simulated performance of the pure software to a custom computing implementation. Synthesis and simulation results indicate:

- trivial reconfigurable area cost of 985 Virtex II CLB slices, 669 Flip-Flops, 22 Block RAMS and 8 multiplier blocks for a basis structure;
- performance improvement (reconfigurable vs. software) of over 5 times for large images;
- larger pictures and longer (polynomial) filters result in even better performance of the reconfigurable design;
- simulations showed the potential of the design to achieve high performance for popular filters used in JPEG2000, when they are factorized in Lifting steps and implemented in the proposed design.

The remainder of the discussion is organized as follows. Section II gives some background information on the DWT and describes the Lifting scheme. Section III discusses the hardware implementation issues followed by the results of the performance analysis, presented in Section IV. Finally, the conclusions are presented in Section V.

II. BACKGROUND

Wavelets introduce a new mathematical concept to decompose a function, say *f(t)*, into sets of other functions referred to as wavelet bases:

$$f(t) = \sum_{\tau,s} c_{\tau,s} \psi_{\tau,s}(t) \tag{1}$$

To have an efficient compression of function *f*, i.e., fewer (nonzero) coefficients $c_{\tau,s}$, it is very important to choose a suitable set of functions $\psi_{\tau,s}(t)$. These functions can be a set of *dilations* (scales) and *translations* of one chosen mother wavelet $\psi(t)$ (see Fig. 1), frequently referred to as *first-generation* (classical) wavelets [9].

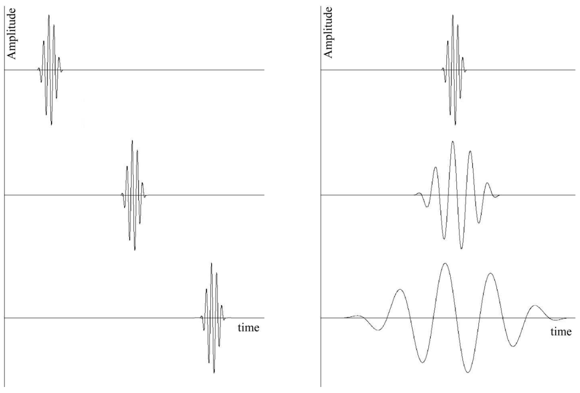

Fig. 1 Translations (left) and dilations (right) of the same prototype (mother) wavelet

This set of functions resembles the basis functions of the Fourier Transform and is defined as:

$$\psi_{\tau,s}(t) = \frac{1}{\sqrt{s}} \psi(\frac{t-\tau}{s}) \tag{2}$$

Where s is the scaling factor and τ is the translation factor. The wavelet coefficients are calculated as follows (*forward wavelet transform*):

$$c_{\tau,s} = \int_{-\infty}^{\infty} f(t) \psi_{\tau,s}(t) d(t) \tag{3}$$

The *inverse wavelet transform*, conversely, uses the computed wavelet coefficients and superimposes them in order to calculate the original data set.

Far more flexible are the *second-generation* wavelets, which are not necessarily translations and dilations of the same prototype function.

A. *The Discrete Wavelet Transform*

In DWT dilation factors are chosen to be powers of 2, therefore the set of dilations and translations of the mother wavelet is defined as:

$$\psi_{j,k}(t) = 2^{-\frac{j}{2}} \psi(2^{-j} t - k) \tag{4}$$

Here *j* is the *scaling* factor and *k* is the *translation* factor. Forward and inverse transforms are then calculated using the following equations:

$$c_{j,k} = \int_{-\infty}^{\infty} f(t) \psi_{j,k}(t) d(t) \tag{5}$$

$$f(t) = \sum_{j,k} c_{j,k} \psi_{j,k}(t) \tag{6}$$

The DWT analyzes the data at different frequencies with different (time) resolutions. This principle is called Multi Resolution Analysis (MRA) and means that we analyze data in different window sizes. When analyzing with a large window, we notice the global behavior of the signal and, conversely, when analyzing with a small window, we focus on its local features. Wavelet functions can be neatly held finite in both time and frequency domains. Therefore they can be used to approximate data with discontinuities or spikes or detect the contours of objects in images [4]. Multi resolution decomposition of a signal into its coarser and finer components is useful for data compression, feature extraction and de-noising.

The DWT can be implemented using different prospects of the transform. For example, the wavelet coefficients can be generated using *dyadic filter banks*, called synthesis filters. The input signal is split into two signals using a lowpass filter *h(t)* and its orthogonal highpass filter *g(t)*. Multiple "scales" are obtained by repeating the filtering process on the lowpass branch outputs only. Another implementation-oriented prospect of the DWT is the *Fast Wavelet Transform*. The DWT is factorized into a product of a few sparse matrices using similarity properties. When these factors are multiplied by a vector, the order of operations reduces, therefore the transform is called "fast". At algorithmic level, however, the fastest ever known realization of DWT is based on the so-called *lifting scheme*.

B. The lifting scheme

One recent, fast implementation of DWT is the Lifting scheme [10], which can be used to construct both first and second-generation wavelets. The idea behind the Lifting scheme is to use half of the data samples to predict the other half and repeat this until all the samples are predicted. The algorithm consists of three simple steps, applied repetitively on the samples: *Split phase*, *Predict phase* and *Update phase*, all illustrated in Fig. 2.

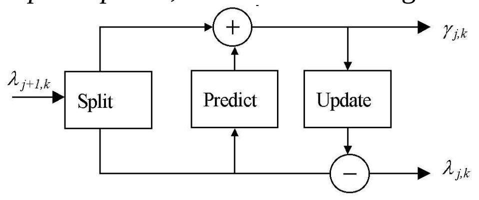

Fig. 2: Split, Predict and Update phases in the lifting scheme (forward transform)

Split phase: Assume that the scheme starts at level 0. We denote the data set as $\lambda_{0,k}$ where k represents the data element and 0 signifies the iteration level 0. In the first stage, the data set is split into two other sets: the even samples $\lambda_{-1,k}$ and the odd samples $\gamma_{-1,k}$ (see Fig. 2). This is also referred to as the *Lazy Wavelet* transform because it does not de-correlate the data, but just sub-samples the signal into even and odd samples. We use negative indices according to the convention that the smaller the data set, the smaller the index:

$$\lambda_{-1,k} = \lambda_{0,2k} \tag{7}$$

$$\gamma_{-1,k} = \lambda_{0,2k+1} \tag{8}$$

Predict Phase (dual lifting): The next step is to use the even sub-set $\lambda_{-1,k}$ to predict the odd sub-set $\gamma_{-1,k}$ using a prediction function $P(\lambda_{-1,k})$. The more correlation presented in the original data, the closer will the predicted value be to the original $\gamma_{-1,k}$. Now, the odd set $\gamma_{-1,k}$ will be replaced by the difference between itself and its predicted value. Thus,

$$\gamma_{-1,k} = \lambda_{-1,k} - \mathrm{P}\,(\lambda_{-1,k}) \tag{9}$$

Different functions can be used for prediction of odd samples. The easiest choice is to predict that an odd sample is just equal to its neighboring even sample. This prediction method is result to the *Haar* wavelet. Obviously, this is an easy but not realistic choice, as there is no reason why the odd samples should be equal to the even ones. Alternatively, second or higher degree interpolation functions can be used for prediction. Depending on the degree of the interpolating function *N*, we can measure failure to predict γ. *N* is referred to as the number of *dual vanishing moments* and defines the degree of the polynomials that can be predicted by the dual wavelet.

Update Phase (primal lifting): In this stage the coefficients $\lambda_{-1,k}$ are lifted with the help of the neighboring wavelet coefficients γ, so that a certain scalar quantity Q, e.g. the mean, is preserved.

$$\mathrm{Q}(\lambda_{-1,k}) = \mathrm{Q}(\lambda_{0,k}) \tag{10}$$

A new operator *U* is introduced that ensures the preservation of this quantity:

$$\lambda_{-1,k} := \lambda_{-1,k} + U(\gamma_{-1,k}) \tag{11}$$

Operator *U* uses a wavelet coefficient of the current level $(\gamma_{j,k})$ to update $\tilde{N}$ even samples of the same level $(\lambda_{j,k})$. $\tilde{N}$ is also known as the number of *real vanishing moments*, not necessary equal to *N*.

Inverse Transform: One of the advantages of the lifting scheme is that the inverse transform is very trivial. The inversion rules are: revert the order of the operations, invert the signs in the lifting steps, and replace the split step by a *Merge step*:

1- Update phase $\lambda_{j,k} = \lambda_{j,k} - U(\gamma_{j,k})$

2- Predict phase $\gamma_{j,k} = \gamma_{j,k} + P(\lambda_{j,k})$

3- Merge phase: $\lambda_{j+1,2k} = \lambda_{j,k} \cup \lambda_{j+1,2k+1} = \gamma_{j,k}$

These three phases are repeatedly applied on the even samples, transforming half of the samples each pass (level), until all samples are transformed.

2-D transform: 2-D transform is performed by applying the 1-D transform algorithm consecutively on the rows and columns of a 2-D signal. Starting from the first iteration level, the 1-D forward transform is first applied to all the rows, and then to all the columns. Subsequently we move to the next iteration level and repeat the afore mentioned steps, and so on .

Advantages of the Lifting Scheme:

1. Lifting scheme is fast: For long filters, Lifting scheme has a complexity of order n/2, compared with a complexity of order n for classical wavelet implementation.
2. All operations within a lifting step can be done entirely in parallel, while the only sequential part is the order of lifting operations.
3. Lifting can be done in-place, therefore an auxiliary memory is not needed. At every summation point the new stream replaces the old one.
4. Lifting allows integer-to-integer transform, while keeping a perfect reconstruction of the original data set.
5. Lifting allows adaptive wavelet transforms. The analysis of a function can start from the coarsest level, followed by data processing at finer levels in the areas of interest.

III. HARDWARE IMPLEMENTATION

The implementation of the lifting algorithm will be explained assuming a signal of length 12 and a polynomial filter with numbers of dual and real vanishing moments, both equal to 4 (*L=12, N=4,* $\tilde{N}=4$).

A. The Predict module

All predict phase calculations are illustrated in Fig. 3. The magnified circle depicts the arithmetic operations involved in the calculation of each γ:

$$\gamma_{j-1} - = (\lambda_{j,k}.F_{i,0}) + (\lambda_{j,k+2}.F_{i,1}) + (\lambda_{j,k+4}.F_{i,2}) + (\lambda_{j,k+6}.F_{i,3})$$

In other words: 4 multiplications, 4 additions/ subtractions and 16 memory accesses (4 times read λ, read lifting coefficients, read γ, and write γ). These operations consume quite many processor cycles when implemented sequentially on a general-purpose processor, which motivated our ultimate goal to speed up these calculations in hardware. In our design we introduce parallel processing and data reusability as depicted in Fig. 4. To meet the requirement for high memory throughput, we break this design problem into four sub-problems and devise a solution to each of them:

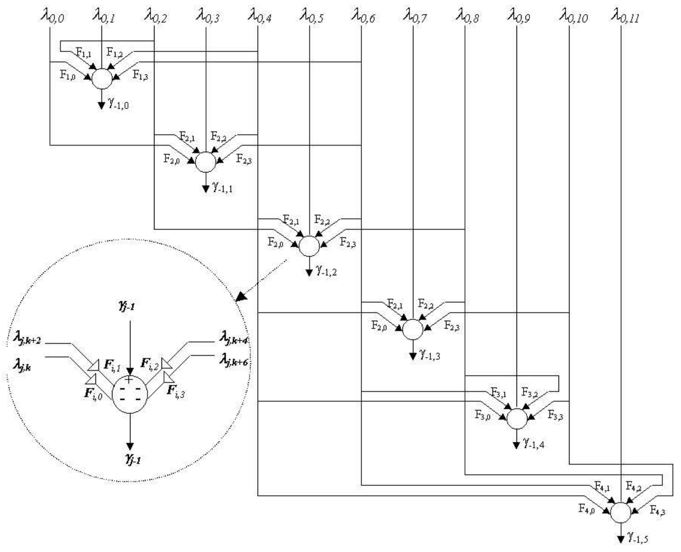

Fig. 3: Calculations of the Predict phase

1. Accessing N λs concurrently
2. Accessing N filter coefficients concurrently
3. Reading input γ concurrently
4. Writing back predicted γ concurrently

Accessing N λs concurrently: From Fig. 3, it can be seen that in the calculations of two consecutive γs, at least three out of four λs are common. This implies that reading only one λ from the memory will be sufficient to calculate the following γ,, provided that λs from the preceding calculations are temporally stored in buffers for reuse. Therefore, we implemented a pipeline of N stages for λ inputs. When being initialized, the pipeline is filled in N cycles. After that, N λs are issued to the unit in parallel (see Fig. 4)

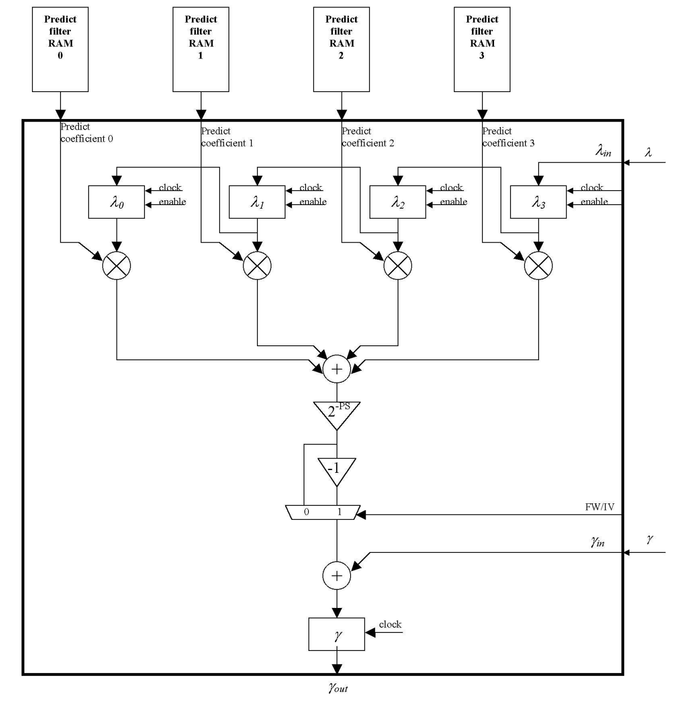

Fig. 4: The Predict module

Accessing N filter coefficients concurrently: This can be accomplished by using a separate bank of RAM for the filter coefficients, as depicted in Fig. 4.

Reading input γ concurrently: For the parallel processing of the unit, γ inputs have to be read simultaneously with λ inputs. This means that two different locations of the image storage area have to be accessed at the same moment. To solve this design problem, we utilize the embedded true dual port RAM blocks of the Xilinx Virtex II FPGA chip. These RAM blocks feature two separate input/output ports, which can be addressed independently from each other. Using these internal dual port RAMs for picture data, γ and λ inputs can be accessed concurrently (see Fig. 4).

Writing back predicted γ concurrently: Reading γ and λ inputs simultaneously, occupies both ports of the dual port picture RAM. To write the predicted γ back into memory within the same system cycle, we designed the picture RAM to operate at a frequency two times higher than the rest of the design. In this case four memory locations per system cycle are addressed.

B. The update module

Fig. 5 illustrates the calculations for the update phase. In each row, one γ updates 4 λs, which implemented in software looks like:

```
for n=0 to $\tilde{N}$
{ $\lambda_{j-1,n}$ += $\gamma_i * L_{i,n}$ }
```

This results in 4 multiplications, 4 additions/subtractions and 16 memory accesses. To synchronize the update with the predict modules, we utilize the same design considerations for both of them. The same problems, regarding memory throughput, have been solved:

1. Accessing $\tilde{N}$ λs concurrently
2. Accessing $\tilde{N}$ filter coefficients concurrently
3. Reading input γ concurrently
4. Writing back $\tilde{N}$ updated λs concurrently

Concurrent access to $\tilde{N}$ filter coefficient is implemented similarly to the prediction module.

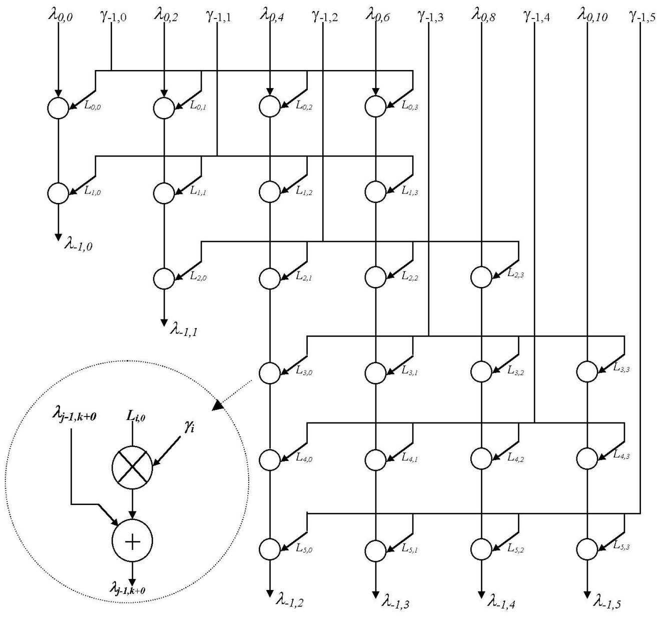

Fig. 5: Calculations of the Update phase

Observing Fig. 5, it can be noticed that the output of each row is the input of the subsequent row, except for at most one output, which has to be written back to the memory. It means that if the outputs of each stage are made available for the inputs of the following stage, all stages lead to at most one value to be written back to the memory, and one value to be read from the memory. The exception here is the first row, which needs 4 memory reads, and the last row that produces 4 outputs to be written back, which have to be done in 4 cycles. This reduces $\tilde{N}$ memory reads and $\tilde{N}$ memory writes to only one read and one write per cycle. Fig. 6 depicts the update module. The pipeline is filled from different sources to accommodate all combinations illustrated in Fig. 5.

To increase the calculation accuracy, the stored filter data is scaled up. The predict and update modules in Fig. 4 and Fig.6, include logic for scaling down the result after each multiplication (illustrated by triangular symbols). The control signal FW/IV switches between summation and subtraction, to make the modules perform both forward and inverse transform.

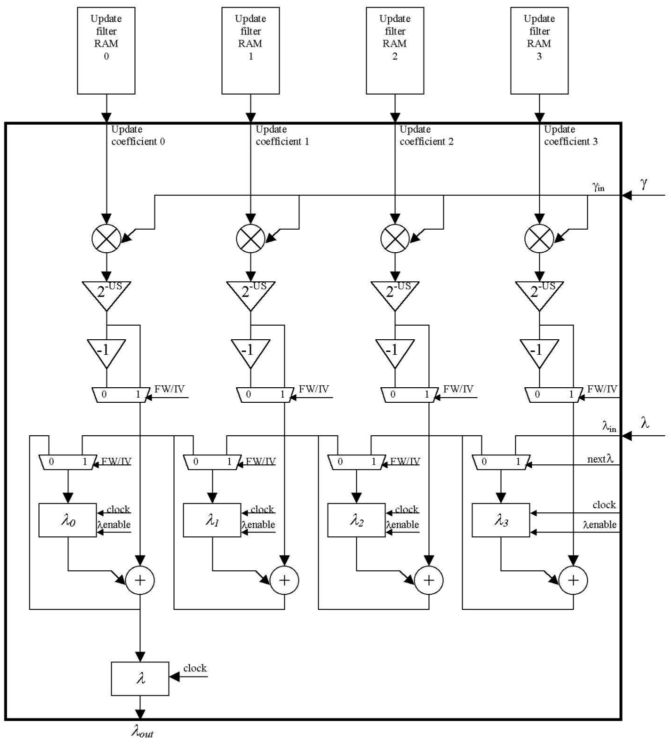

Fig. 6: The Update module

C. Parallel operation of Predict and Update modules

It was explained how the predict and update modules calculate the outputs in one cycle. However, parallel operation of both modules leads to 6 memory operations per cycle (for predict module, λ and γ inputs and γ output, and for update module, λ and γ inputs and λ output). The dual port memory is accessed twice per system cycle, accommodating a maximum of 4 memory accesses a cycle. Therefore, parallel operation of both predict and update module, exceeds the memory bandwidth. In the forward transform, the three required memory accesses for the predict module can be directly accommodated from the picture RAM. Also the output of the update module can be directly written back to the picture. Further, we explain how data are provided to the inputs of the Update module.

Providing data to γ input of the update module: It can be seen in Fig. 2 that the γ input of the update module can be fed with the output of the predict module. Introducing a First-In-First-Out (FIFO) buffer between the output of predict module and the γ input of the update module, absorbs the unequal delivery and consumption rate of data at the beginning and the end of the predict and update phases (see Fig. 7).

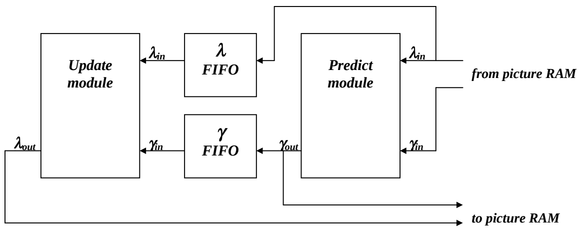

Fig. 7: Synchronizing FIFO buffers for forward transform

Providing data to λ input of the update module: Fig. 2 shows that the update phase uses the same λs as the predict phase, but at a different time. Consequently, it is not possible to fill the λ pipelines of both predict and update modules with the output of the picture RAM. Placing a FIFO buffer before the λ input of the update module is the answer to this timing problem. This buffer, in fact, solves the synchronization problems and allows the update module to reuse the λs read by the predict module (see Fig. 7). In inverse transform the FIFOs provide data to the inputs of the predict module. Fig. 8 illustrates the schematic view of the design.

D. Hardware resource usage

As we mentioned earlier, we have synthesized the design for the Xilinx Virtex II FPGA technology. Table I presents the hardware resource usage of the unit for a test picture of 64x32 pixels (8 bits gray-scale) and 4-4 polynomial filter. Except for the number of RAM blocks, the values in Table I also apply for larger picture dimensions.

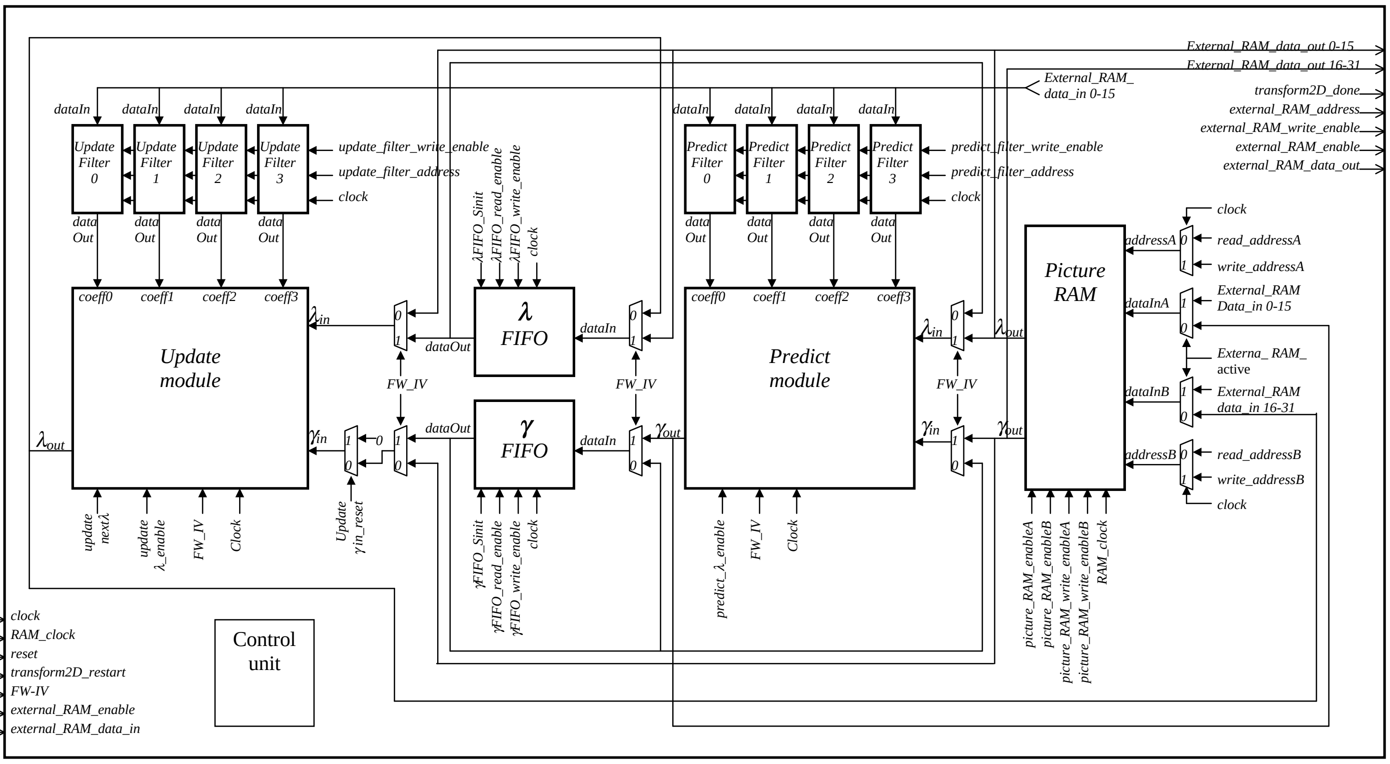

Fig. 8: Top level organization of the hardware DWT unit.

TABLE I

HARDWARE RESOURCE USAGE OF THE DESIGN

Target Device	Xilinx x2v1000
Clock frequency	> 5x10^7 Hz
Number of Slices	985
Total Number 4 input LUTs	1637
Number of Flip-Flops	669
Number of Block RAMs	22
Number of MULT18X18s	8

IV. PERFORMANCE ANALYSIS

In this section we explain the performance analysis framework and present the result of simulations carried out on different images and filters to analyze the performance of the module.

A. Performance analysis benchmark

To measure the performance improvement due to the introduction of the new functional unit, the execution time of a pure software implementation of the Fast Lifting DWT algorithm was compared to the execution time when using the hardware accelerator. Using Liftpack [1] as software implementation and Sim-outorder of the Simplescalar toolset [3], a benchmark was setup to define the software execution time. [7] and [8] explain the performance analysis in detail. The performance measurement metric is given by:

$$Speedup = \frac{SW\ exec\ time}{HW\ exec\ time}, \qquad (12)$$

where *SW exec time* and *HW exec time* denote software and hardware (execution) times spent to execute the benchmark algorithm in software and in hardware respectively. Denoting *TNEC* as the Total Number of Execution Cycles, assuming a general purpose processor running at 1 GHZ (10^{9}Hz) and DWT unit running at 5x10^7 Hz (see Table I), we define:

$$SW\ exec\ time = \frac{TNEC_{SW}}{10^9} \quad ,[\mathrm{sec}] \qquad (13)$$

$$HW\ exec\ time = HW\ tr\ time + DTT \quad ,[\mathrm{sec}] \qquad (14)$$

$$HW\ tr\ time = \frac{TNEC_{HW}}{5\times10^7} \quad ,[\mathrm{sec}] \qquad (15)$$

where *HW tr time* is the time for the hardware DWT transform. DTT denotes *data transfer time*, which is the time for sending data to and from the hardware module.

B. *Performance analysis of different polynomial filters*

A number of simulations were performed in order to compare the performance gains due to hardware acceleration for different configurations of picture size and filter types. In this section, the performance gain for different polynomial filters are compared. Table II summarizes the results of the simulations for a constant picture size of 352 x 288 and Fig.9 illustrates the performance gains graphically. The substantial performance increase for higher polynomial degrees is due to the almost constant hardware execution time, while the software execution time increases dramatically with the degree of the filters. Once the pipelines of the module are filled (Fig. 4 and 6), the design generates two outputs per clock cycle (one λ and one γ) irrespective of the length of the filters. This is one of the primary reasons for superior performance of the design.

TABLE II

PERFORMANCE ANALYSIS - DIFFERENT DEGREES OF POLYNOMIAL FILTERS

Filter of polynomial degrees	2-2	4-4	8-8
SW execution time (µsec)	9831	13395	22061
HW execution time (µsec)	3210	3370	3670
Performance ratio (HW vs. SW)	**3.06**	**3.97**	**6.01**

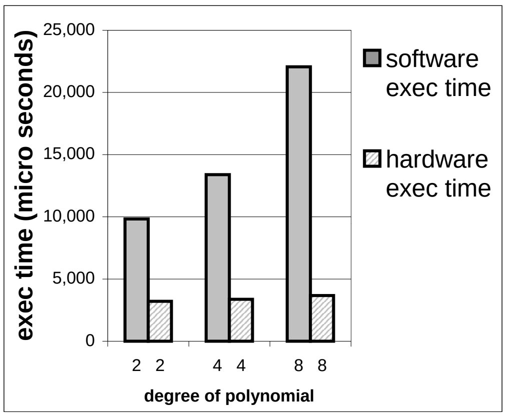

Fig. 9: Performance gain- different degrees of polynomial filters

C. *Performance analysis for different picture sizes*

Filling the pipelines at the start of each 1-D transform (a row or a column) leads to cycles in which no output is generated. Furthermore the parallel execution of the predict and update modules cannot start immediately at the beginning of the 1-D transform, but after number of cycles, when both of the inputs become available. The relative effect of these non-productive cycles attenuates as length of the 1-D signal (picture dimension) increases, yielding a better performance gain for larger pictures. Table III and Fig. 10 compare the results of simulations on different picture sizes with the same polynomial filter (4-4).

TABLE III

PERFORMANCE ANALYSIS – DIFFERENT PICTURE SIZES

Picture format	176x144	352x288	720x560
SW execution time (µsec)	3315	13395	63301
HW execution time (µsec)	962	3370	12577
Performance ratio (HW vs. SW)	**3.44**	**3.97**	**5.03**

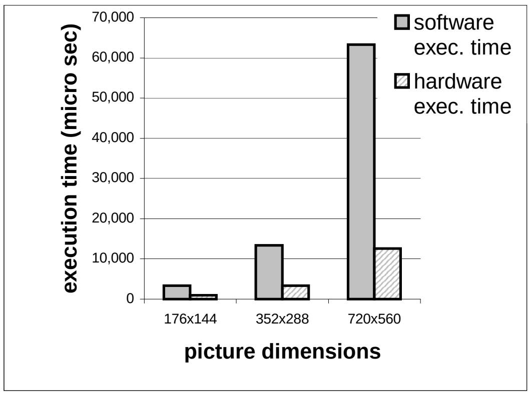

Fig. 10:Performance improvement for different picture dimensions

D. *Performance analysis of other filters*

The proposed hardware unit has been based on polynomial filters. However, it is possible to implement other filters, when they are factorized into Lifting steps, with minor modifications in the design. We implemented two popular filters, namely Le Gall 5-3 and Daubechies 9-7 in software, and calculated the expected performance gain if our hardware unit would have been used. For the hardware accelerated performance, the 2-2 polynomial filter was used as estimation. The results (Table IV and Fig. 11) show that using hardware acceleration based on the proposed design, an estimated performance enhancement of factor 3.69 might be achievable for Le Gall 5-3 filter. This estimated factor increases to more than 11 for more computationally intensive Daubechies 9-7 filter. This analysis shows the great potential of the design for practical applications as JPEG2000 and MPEG-4 in which these two filters are used.

TABLE IV

PERFORMANCE ANALYSIS – DIFFERENT FILTER TYPES

Filter type	Polynomial 2-2	LeGall 5-3	Daubechies 9-7
SW execution time (μsec)	8833	11860	35990
HW execution time (μsec)	3210	3210	3210
Estimated performance ratio HW vs. SW	**2.75**	**3.69**	**11.21**

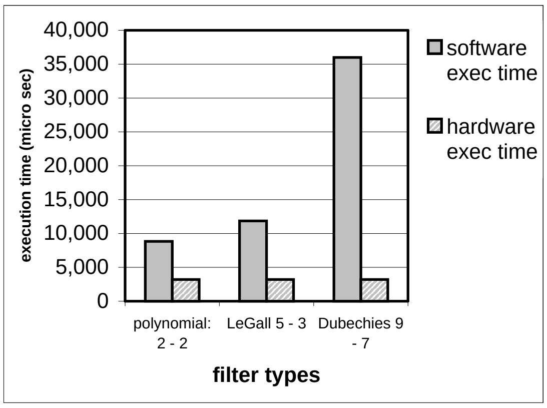

Fig. 11: Estimated acceleration for some popular filters

E. Reconfigurable computing environment

The proposed hardware unit is intended to operate in a custom-computing platform. The idea behind is that a reconfigurable hardware co-exists with a core general purpose processor [6]. More precisely we target to the novel reconfigurable multimedia processors called MOLEN [2]. The concept of these processors is to extend general-purpose architectures with application specific co-processing (DWT in our case) mapped on FPGAs and by means of microcode to improve FPGA setting and execution.

V. CONCLUSIONS

In this paper we introduced a new hardware unit to accelerate the Discrete Wavelet Transform. The unit is based on the Lifting scheme algorithm. We explained how different parts of the algorithm can be executed in parallel, if implemented in hardware. Subsequently we presented the results of performance analysis of the hardware module. We compared these results to a pure software implementation and concluded that the proposed module could be beneficially used to accelerate the performance of the Discrete Wavelet Transform in applications using it, like JPEG2000 and MPEG-4.

VI. ACKNOWLEDGMENTS

This research has been supported by PROGRESS, the embedded systems research program of the Dutch organization for scientific research NWO, the Dutch Ministry of Economic Affairs, and the Technology Foundation STW (project AES.5021).

REFERENCES

[1] G. Fernandez, S. Periaswamy, and W. Sweldens, LIFTPACK: A software package for wavelet transforms using lifting. *Wavelet Applications in Signal and Image Processing IV,* pages 396-408. Proc. SPIE 2825, 1996, http://www.cse.sc.edu/~fernande/liftpack

[2] S.Vassiliadis, S. Wong, and S. Cotofana. The MOLEN ρμ-coded processor. *In 11th International Conference on Field Programmable Logic and Applications (FPL)*, 2001

[3] Doug Burger, Todd M. Austin. The SimpleScalar Tool Set, Version 2.0. Technical Report CS-TR-1997-1342, University of Wisconsin-Madison, 1997. http://www.simplescalar.com

[4] W. Press et al., Numerical recipes in Fortran, Cambridge university press, New York, 1992

[5] Geert Uytterhoven. Wavelets: software and applications, PhD thesis, April 1999, Catholic University of Leuven.

[6] Stephan Wong, Sorin Cotofana, and Stamatis Vassiliadis. Coarse Reconfigurable Multimedia Unit Extension, *Proceedings of the 9th Euromicro Workshop on Parallel and Distributed Processing PDP 2001*

[7] B.Zafarifar. Micro-codable discrete wavelet transform, M.Sc. thesis, Delft University of Technology, 1-68340-28 (2002)-03

[8] P. Shrestha. MIPS augmented with Wavelet Transform, Performance Analysis, M.Sc. Thesis, Delft University of Technology, 1-68340-28(2002)-02

[9] C. M. Brislawn. Classification of nonexpansive symmetric extension transforms for multirate filter banks. *Applied and Comp. Harmonic Analysis*, 3:337-357, 1996

[10] W. Sweldens. The lifting scheme: a custom-design construction of biorthogonal wavelets. *Applied and Comp. Harmonic Analysis*, 3(2):186-200, 1996

[11] XILINX. Virtex-II Platform FPGA Handbook, December 2000.