Microarchitectural Extension for Lifting-based DWT

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)15-27-87364 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, we investigate a novel microarchitectural extension for the DWT based on the lifting scheme. A new fast lifting DWT (FLWT) instruction is introduced as an ISA Extension of a MIPS architecture. Simulations have been carried out by a cycle accurate simulator (SimpleScalar). As benchmark software, we have used a modified version of the package LIFTPACK, optimized for integer arithmetic. A microcodable DWT unit has been implemented on a reconfigurable hardware platform - namely the Xilinx FPGA VIRTEX II. In order to accelerate the transform process, the hardware module utilizes pipelining, data reusability, data parallelism and some specific features of the Xilinx FPGAs. We have used the original synthesis tools of the FPGA vendor to get realistic data for the performance of the unit. Simulation results indicate a speedup of over 4 times versus the pure software implementation, assuming clock frequencies of 50 MHz for the DWT module and 1 GHz for the General Purpose MIPS. An important advantage of the module is that for higher degrees of the filter polynomial, its speed up versus the pure software implementation would be even higher. It is also noted that the speedup increases when the dimensions of the processed pictures grow. More dramatic improvements in performance can be achieved, since the design can be scaled-up to utilize higher degrees of parallelism. The introduced microarchitecture can be utilized by wavelet based encoding tools and standards like JPEG 2000, MPEG-4, etc.**

Keywords— **Data compression, DWT, Lifting scheme, Microarchitecture, ISA Extension**

I. Introduction

Due to its advantageous features, the Discrete Wavelet Transform (DWT) has emerged to become a basic encoding algorithm for recent data compression standards. During the last decade intensive research efforts have been spent both on the theoretical foundations of DWT and on its practical implementation in modern signal and image processing systems. Some of the advantages of wavelets are:

- Good decorrelating behavior which can be applied for efficient compaction of data.
- Localization both in space(time) and scale (frequency) domains. Hence wavelets can easily detect local features in a signal.
- Wavelets are based on multi-resolution analysis, which allows analyzing a signal at different resolution levels.
- Wavelets are smooth, which can be characterized by their number of vanishing moments (to be explained later). The higher the number of vanishing moments, the better smooth signals can be approximated in a wavelet basis.
- Fast and stable algorithms to calculate the Discrete Wavelet Transform (DWT) and its inverse are already available.

On the other hand, regarding the required computational time, a pure software implementation of the DWT is recognized to be a substantial performance bottleneck in systems utilizing it. Therefore, its acceleration is a key research issue. At algorithmic level, the so called *lifting scheme* [7] represents an implementation of the DWT, recognized to be the fastest ever known.

In this paper we evaluate a hardware acceleration of the lifting scheme, using a microcodable Fast Lifting Wavelet (FLWT) co-processing unit coupled with a General Purpose Processor (MIPS in this case). As a benchmark software we utilize a modified version of LiftPack [4], a package for lifting based wavelet transforms. The extended microarchitecture is simulated by Sim-Out-Order, a cycle accurate simulator from the SimpleScalar tool set [3]. An FPGA prototype of the FLWT unit has been designed [10] and synthesized with the recent (Virtex II) technology of Xilinx [9] . The timing parameters, obtained from the synthesis, are used for the microarchitectural simulations. Further, we implement Polinomial, Daubechies 9-7, and Reversible Le Gall 5-3 filters with bi-orthogonal wavelets and analyze the hardware vs. software (HW/SW) speedups by changing various parameters.

Assuming clock frequency of 1GHz for the General Purpose Processor and 50 Mhz for the FLWT unit, simulation results indicate:

• HW/SW speedups of 3.5, 4.0, and 5.0 for picture formats of 176×144, 352×288, and 720×560 respectively.
• HW/SW speedups of 3, 4, and 6 for filters with Polinomial degrees of 2-2, 4-4, and 8-8 respectively.
• HW/SW speedups estimated to be 3, 3.7, and 11 for polinomial, Reversible Le Gall 5-3, and Daubechies 9-7 bi-orthogonal filters respectively.

The remainder of the discussion is organized as follows. Section II gives some background information on DWT and its possible implementations. In Section III, the lifting scheme is discussed. Section IV describes the investigated microarchitectural extension and some modeling considerations. Simulation results are reported in Section V. Finally, the conclusions are represented in Section VI.

II. Background

Wavelets, literally meaning small waves, are mathematical concepts for decomposing a function, say f, into sets of other functions known as wavelet bases- $\Psi_{a,b}(t)$.

$$f = \sum_{t} C_{a,b} \cdot \Psi_{a,b}(t) \tag{1}$$

To have an efficient representation of the signal f using only a few coefficients $C_{a,b}$, it is very important to use a suitable family of functions $\Psi_{a,b}$. The functions $\Psi_{a,b}$ should match the features of the data we want to represent. Time-limited signals (space-limited in case of pictures) can be represented efficiently using a basis of block functions (Dirac delta functions), but these block signals do not efficiently handle frequency limitations. Band-limited signals can be represented efficiently using a Fourier basis but Sines and Cosines are not limited in time. Wavelet functions, as a compromise between the pure time-limited and band-limited basis functions, combine the best of both.

In order to get the variable time-frequency localization *(resolution)*, a wavelet called *mother wavelet* or *prototype function* $\Psi(t)$ is defined as shown in Figure 1. Basis functions $\Psi_{a,b}(t)$ are calculated as scaled and translated versions of the prototype:

$$\Psi_{a,b}(t) = \frac{1}{\sqrt{b}} \cdot \Psi(\frac{t-a}{b}), \tag{2}$$

where b is the scaling coefficient and a is the translating coefficient.

The **Discrete Wavelet Transform (DWT)** with resolution level j (scale $b = 2^j$) at time k is defined as:

$$\Psi_{j,k}(t) = 2^{-\frac{j}{2}} \cdot \Psi(2^{-j} \cdot t - k) \tag{3}$$

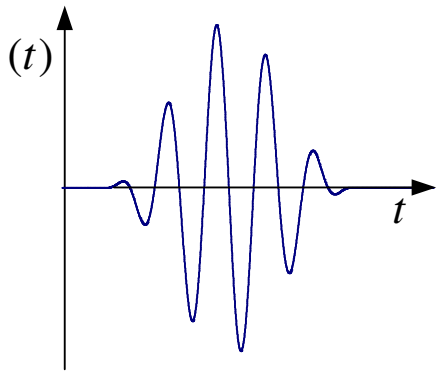

Fig. 1. Wavelet Prototype Function - an example

Considering Equations (1) and (3), any signal can be represented as the sum of a set of wavelet coefficients at an infinite number of scales:

$$f(t) = \sum_{j,k} C_{j,k} \cdot \Psi_{j,k}(t) \tag{4}$$

$$C_{j,k} = \int_{-\infty}^{+\infty} f(t) \cdot \Psi_{j,k}(t) dt \tag{5}$$

This equation resembles the Fourier Transform and is called the *classic wavelet transform* [1].

Wavelets in image compression. Some of the features of wavelets, that make them very successful and widely implemeted in recent image compression algorithms are:
• Wavelets provide high compression ratios: in terms of visual quality they perform much better than competing technologies like DCT.
• The wavelet transforms are symmetric: both the forward and the inverse transform have the same complexity, allowing fast compression and decompression.
• Multi-resolution signal analysis allows progressive transmission and zooming, without the need for extra storage.
• Wavelets can be used for various image-processing operations. The possibility to combine image processing and compression is a very appealing factor.

DWT implementations. DWT can be implemented using different prospects of the transform. For example, the wavelet transform coefficients can be generated using 2 channel *filter banks* called synthesis filters. The input signal is split into two signals using a lowpass filter *h(t)* and its orthogonal or bi-orthogonal highpass filter *g(t)*. Multiple levels or "scales" of the wavelet transform are made by repeating the filtering and decimation process on the lowpass branch outputs

only. The process is typically carried out for a finite number of scales, resulting the wavelet coefficients.

Another implementation-oriented prospect of the DWT is the *Fast Wavelet Transform*. In this case, the DWT can be factorized into a product of a few sparse matrices using similarity properties. When these factors are applied to the multiplication with a vector, the order of operations reduces, thus the transform is called "fast".

At algorithmic level, however, the fastest ever known DWT implementation is based on the so called *the lifting scheme*. We describe this new prospect of the DWT in the Section to follow .

III. The Lifting Scheme

The Lifting scheme is an efficient implementation of a wavelet transform algorithm. It was primarily developed as a method to improve wavelet transform, and then it was extended to a generic method to create so-called second-generation wavelets (i.e. wavelets which do not necessarily use the same function prototype at different levels). Second-generation wavelets are much more flexible and powerful than first generation wavelets [7]. In [2], it is shown that any discrete wavelet transform can be obtained with a finite number of lifting steps. The lifting scheme is an implementation of the filtering operations at each level. The algorithm can be described in three phases, namely: *Split phase, Predict phase* and *Update phase*, as illustrated in Figure 2.

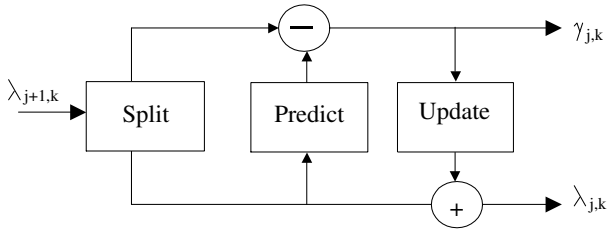

Fig. 2. The Lifting Scheme

Assume the scheme starts at data set of $\lambda_{0,k}$ where k represents the data element and zero signifies the original data level:

Split Phase. In the first stage, the data set $\lambda_{0,k}$ is split into two other sets: the $\lambda_{-1,k}$ and the $\gamma_{-1,k}$. The negative indices have been used according to the convention that the smaller the data set, the smaller the index. The splitting is done by separating the set of even samples and the odd samples. 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.

Predict Phase or Dual Lifting. The next step is to use the $\lambda_{-1,k}$ subset to predict the $\gamma_{-1,k}$ subset with the use of prediction function $P(\lambda_{-1,k})$. The more correlation is presented in the original data, the closer will the predicted value be to the original $\lambda_{-1,k}$. Now, the set $\gamma_{-1,k}$ will be represented by:

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

The prediction functions can be broadly divided into *piecewise linear* (of order 2) and non-linear or i*nterpolating subdivision*:

Piecewise linear model - the odd samples are predicted as the average of its two (even) neighbors, $\lambda_{-1,k}$ and $\lambda_{0,k+1}$, which is given by:

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

Interpolating subdivision - this model uses the same basic idea as piecewise linear but uses 2 or more neighbors to either side.

Depending on the order of subdivision (interpolation), denoted by *N*, the wavelet coefficients can measure failure to predict. For instance, *N =2* indicates the failure to be linear and *N=4* measures the failure to be cubic. *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 or Primal Lifting. In this stage the coefficient $\lambda_{-1,k}$ is lifted with the help of the neighboring wavelet coefficients so that a certain scalar quantity Q (e.g.,the mean), is preserved.

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

Therefore, a new (*update*) operator U is applied:

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

In this phase, a *scaling function* is calculated from the previously calculated wavelet coefficients to maintain some properties among all the λ coefficients throughout every level. The order of this function will be some even value $\tilde{N}$ called real vanishing moment, not necessary equal to *N*.

Inverse Transform. The inverse transform of the lifting scheme is just the reverse data flow in the setup of forward transform with small changes like switching additions and subtractions and also switching divisions and multiplications:

1. Update: $\lambda_{j,k} = \lambda_{j,k} - U(\gamma_{j,k})$
2. Predict: $\gamma_{j,k} = \gamma_{j,k} + P(\lambda_{j,k})$
3. Merge: $\lambda_{j+1,2k} = \lambda_{j,k} \bigcup \lambda_{j+1,2k+1} = \gamma_{j,k}$.

Advantages of the lifting scheme, compared to the classical filter bank algorithm:

1. Lifting leads to a speedup, therefore it is also referred to as *fast lifting wavelet transform* (FLWT).

2. All operations within a lifting step can be done entirely parallel while the only sequential part is the order of lifting operations.
3. Lifting allows adaptive wavelet transforms.
4. Lifting can be done in-place, which means auxiliary memory is not needed. The new stream at every summation point replaces the old one.
5. It is easy to build non-linear wavelet transform by Lifting, for example, integer-to-integer transform.

IV. Modeling the Extended Microarchitecture

In this section we will briefly discuss the hardware platform considerations for a time-effective implementation of the FLWT. We will also present some modelling issues, regarding the proposed simulation scheme.

In order to support the new operations imposed by FLWT and to preserve the flexibility of the software implementation of the wavelet transform, we decided to extend a generic MIPS Instruction Set Architecture (ISA) with one new instruction. In this paper we refer to the new instruction as FLWT and assume that it is supported by a hardware unit, implemented on a reconfigurable platform. Some of the general design considerations of this unit are:

- *Integer to integer transform.* Integer arithmetic is suitable for fast implementations of the discrete wavelet transform at low hardware complexity.
- *Separate control* circuitry, utilizing microcode.
- *Internal pipelining* to exploit *parallel processing* and *data reusability.*
- *FPGA-based implementation* to obtain realistic performance evaluation and to verify the design.

Since hardware implementation details (extensively described in [10]) are outside the scope of this paper, we will proceed to describe the simulation model of the FLWT ISA extension (more details can be found in [6]). A general view of a General Purpose Processor (GPP), augmented with a FLWT is depicted in Figure 3.

The GPP model. In our simulation framework the GPP is a generic MIPS processor, described in the SimpleScalar simulator environment. The new functional unit "FLWT" was added in the resource pool of Sim-Outorder, along with 4 integer ALUs, 1 integer MULT/DIV unit, 4 Floating Point (FP) adders, 1 FP MULT/DIV unit and 2 memory ports. In order not to modify the compiler, we used the instruction annotation mechanism (provided by the SimpleScalar ISA) and in essence extended the ISA with the new

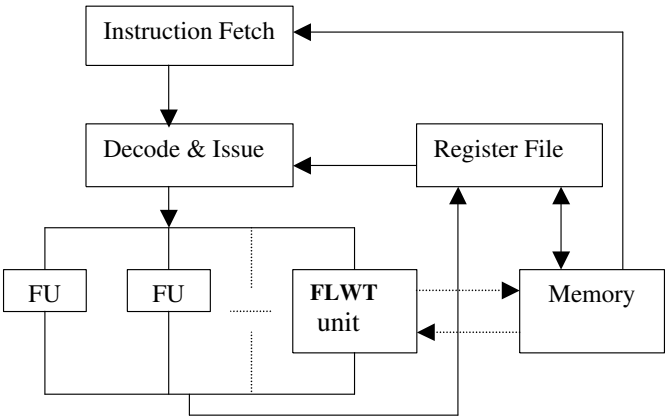

Fig. 3. GPP augmented with FLWT unit

FLWT instruction.

Memory simulation. In SimpleScalar, the cache configuration can be represented in the order of number of blocks, block size and associatively as 128:32:4 and 1024:64:4 for 1st level and 2nd level cache respectively. While reading data, the latency of a hit is 1 cycle. If it is a miss, then the miss penalty is 6 cycles for 1st level cache and 32 cycles for 2nd level cache. The total load latency was calculated as the summation of individual load latencies. After the transform operation, the data was written back to the host memory through the Write Port of SimpleScalar. The write latency is 1 cycle. Both Read and Write Ports are used to transfer data between the FLWT unit and the host memory, enabling read-transform-write pipelining in the unit.

V. Simulation and Results

In order to determine the performance gain provided by the FLWT unit, we compiled both annotated and non-annotated versions of the benchmark source code using tools from the SimpleScalar tool set (Version 3.0). Then we executed both versions of benchmarks on the modified Sim-outorder, which supported the new instructions and contained the FLWT unit.

A. Simulation Framework

Benchmark modifications. Two major modifications of the benchmark software (LiftPack) have been made, beforehand.

- *Integer to Integer Transform.* The original software has been modified towards mapping integers to integers. At the expense of introducing non-linearity in the transform, the Predict and Update filter coefficients have been scaled and rounded to integers. Integer to integer lifting-based DWT is fully reversible and

allows a perfect reconstruction of the original image.
• *Two extra filters* (besides the polinomial) have been implemented to build a realistic application environment. We chose the Le Gall 5-3 and Daubechies 9-7 filters, because they have been approved and included in the normative part of the JPEG 2000 standard[5].

Performance evaluation. Our main performance metric is the speedup of FLWT operation, which is given as the ratio between the software execution time (T_{SW}^{EX}) and hardware execution time (T_{HW}^{EX}).

$$Speedup = \frac{T_{SW}^{EX}}{T_{HW}^{EX}} \quad (6)$$

Software execution time is the total amount of time spent to execute the FLWT operation, without considering the time spent by software for user interfaces and file handling procedures. The *Total Number of Execution Cycles* (TNEC) for actual software implementation is calculated as:

$$TNEC_{SW} = TNEC_{orig} - TNEC_{an}^{0}, [cycles] \quad (7)$$

where $TNEC_{orig}$ is TNEC when the original (non-annotated) source code of LiftPack is run; $TNEC_{an}^{0}$ denotes TNEC when the source code is annotated, to support the new FLWT instruction and the latency of this instruction is set to 0. Using Equation (7), we refine the TNEC spent only by the pure DWT algorithm and ignore all other additional operations in the benchmark software (e.g., operating system calls).

The MIPS processor was assumed to run at 1 GHz, thus the software execution time is;
$T_{SW}^{EX} = \frac{TNEC_{SW}}{10^9}$, [sec]

Hardware execution time is the total time taken by FLWT unit to perform the transform, including the time required to load data into the unit and write it back to the memory. The transform execution time was obtained from the realistic hardware model built in Xillinx FPGA (Virtex II family), which operates at clock frequency of 50 MHz. Regarding data transfer time (DTT_{HW}), we consider two hardware limitations:
• Data transfer time due to memory organization (DTT_{MEM}).
• Data transfer time defined by the maximum operating frequency of the FPGA implementation (DTT_{FPGA}).
From these data transfer times, we consider the longer one:
$DTT_{HW} = MAX(DTT_{MEM}, DTT_{FPGA})$
and the hardware execution time is:
$T_{HW}^{EX} = \frac{TNEC_{HW}}{5\times10^6} + DTT_{HW}$, [sec]

B. Simulation Results

The simulation results were obtained upon executing the benchmark software (LiftPack) on the cycle accurate simulator Sim-Outorder. Due to some implementation issues of the simulator, the results on each execution may deviate in the range of maximum ±2%. For all experiments we assume that the GPP is running at 1 GHz and that the clock frequency of the FPGA implementation of the FLWT unit is 50 MHz.

The performance analysis results obtained from Liftpack are presented in Table I. With the application of the FLWT unit, the execution time of wavelet transform becomes 3.5 times faster for picture size of 176×144, 4 times - for 352×288 and 5 times - for 720×560. The rise in speedup with the picture dimension is because of more efficient use of the pipelines due to longer data elements during 1-D transform. All simulations were performed with the polynomial filter degree of 4-4. A graphical representation of the comparison between hardware execution time and software execution time is shown in Figure 4.

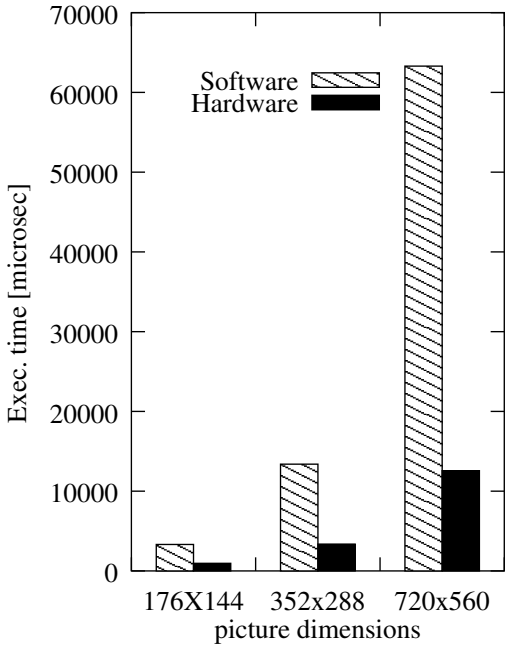

Fig. 4. Influence of picture dimensions on performance

Experiments indicate that the speedup increases when the polynomial degrees of the filter increase. Table II presents the simulation results carried out with different degrees of the polinomial filter from Liftpack. All Experiments were made for a single picture size of

TABLE I

PERFORMANCE ANALYSIS - DIFFERENT PICTURE SIZES

Picture size	176×144	352×288	720×560
TNEC hardware	46 000	160 000	586 000
Time to transform in HW (μsec)	920	3 200	11 720
Pictures per second in HW	1 087	313	85
HW data transfer limit (μsec)	42	170	673
TNEC software	3 314 925	13 395 008	63 300 925
Software Execution Time (μsec)	3 315	13 395	63 301
Hardware Execution Time (μsec)	962	3 370	12 577
Speedup HW vs. SW	**3,5**	**4,0**	**5,0**

TABLE II

PERFORMANCE ANALYSIS - DIFFERENT DEGREES OF POLINOMIAL FILTERS

Filter of polinomial degree	2-2	4-4	8-8
TNEC hardware	152 000	160 000	175 000
Time to transform in HW (μsec)	3 040	3 200	3 500
Pictures per second in HW	329	313	286
HW data transfer limit (μsec)	170	170	170
TNEC software	9 831 292	13 395 008	22 061 284
Software Execution Time (μsec)	9 831	13 395	22 061
Hardware Execution Time (μsec)	3 210	3 370	3 670
Speedup HW vs. SW	**3,0**	**4,0**	**6,0**

352×288 pixels. The speedups for filter degrees of 2×2, 4×4, and 8×8 are 3, 4 and 6 times respectively. These results are visualized in Figure 5.

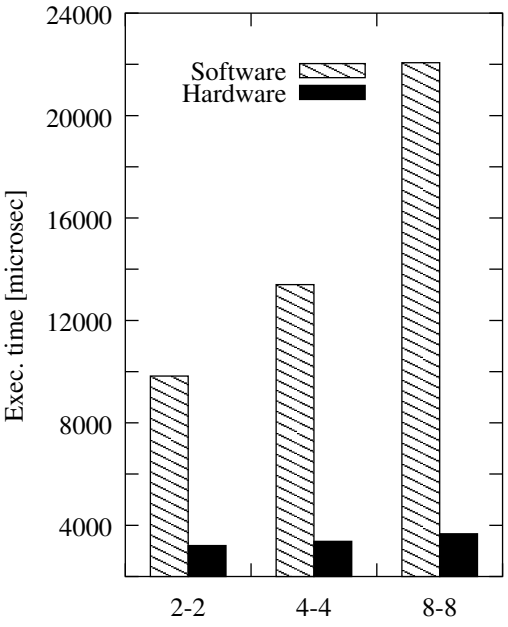

Fig. 5. Performance for different polinomial filter degrees

In Table III, we present simulation results for reversible and irreversible wavelet transform using polynomial filter, Le Gall 3-5 filter and Daubechies 9-7 filters. We made an estimation of the hardware transform time on the basis of Liftpack with polynomial filer 2-2 and carried out the analysis for a 352×288 picture. It was found that speedup of Daubechies 9-7, Le Gall 5-3 and Liftpack with 2-2 polynomial was 11, 3,7 and 2,8 times respectively (Figure 6).

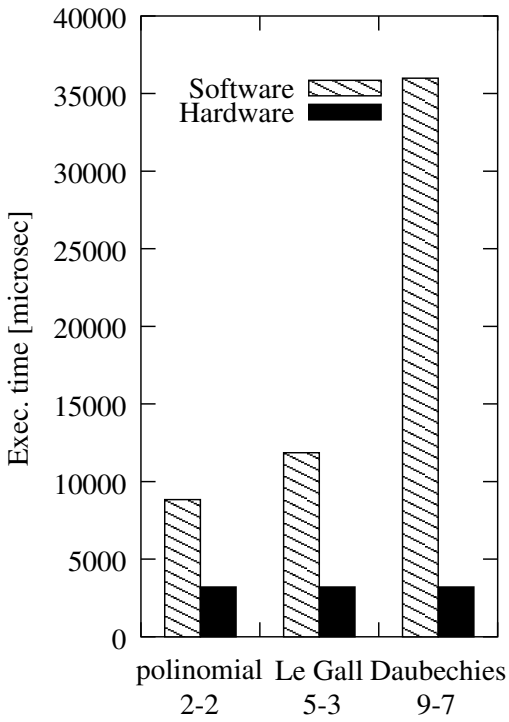

Fig. 6. Performance comparison of Polynomial, Le Gall 5-3 and Daubechies 9-7 filters

TABLE III
PERFORMANCE ANALYSIS - DIFFERENT FILTER TYPES

Filter type	polynomial: 2-2	Le Gall 5-3	Daubechies 9-7
Estimated TNEC hardware	152 000	160 000	175 000
Time to transform in HW (μsec)	3 040	3 040	3 040
Pictures per second in HW	329	329	329
HW data transfer limit (μsec)	170	170	170
TNEC software	8 833 192	11 860 033	35 990 178
Software Execution Time (μsec)	8 833	11 860	35 990
Hardware Execution Time (μsec)	3 210	3 210	3 210
Speedup HW vs. SW	**2,8**	**3,7**	**11,0**

To get a quantitative measure of the effect of the memory latencies on the hardware execution time, we analyze 2 cases:

Case 1: The picture data are read from memory for each execution of the 1-D transform and written back after the transform. This requires less internal memory, equal to the length of a line. For instance, the picture size of 352×288 requires only 352 storage elements. However, data have to be read each time for the transform operation.

Case 2: All picture data are read at the beginning of the transform and written back after the completion of the 2-D transform. This requires a sufficiently large internal memory to store the full picture data but less memory transfers than in case 1.

The simulation data for different picture sizes and their analysis is given in Table IV. *The total number of pixels to be transferred* was obtained from the Simplescalar simulator and reflects the number of times the FLWT unit accesses the host memory, during the total transform operation. *The total read/write latency (processor cycles)* indicate the number of cycles that the external RAM takes to transfer the data to and from the hardware module. *The total read/write latency time by processor* is the time that the external RAM takes to transfer the data to and from the hardware module.

The results indicate that the latencies for read/write operations for *case 1* are 6 times more for picture size 352×288 and 12 times more for picture size 720×560 in comparison to Case 2. The reason is that the FLWT algorithm operates multiple times on the same pixel in different levels, according to its multi resolution scheme. *Case 1* requires reading 270 270 pixels for a picture format of 352×288 pixels, while *case 2* requires only 101 376 pixels for the picture of the same format. In addition, the more data are read, the equal data have to be written back, hence resulting more writing latency.

Another major factor for increasing the latency in *case 1* is the inherent cache misses due to accessing the sparse memory locations, which cannot be held in a cache memory. Data of consecutive rows of an image are generally aligned in a linear array in the external memory. In *case 2*, these data are read sequentially resulting in less frequent misses. But, in *case 1* data are sequentially accessed only during the first level horizontal transform, while in the first level column transform the distance between two data elements is equal to the length of a row of the picture. With the increase in the level of transform the distance between the data to be read grows and thus the reading latency also increases.

The effect of reading and writing latencies on the hardware execution time is shown graphically in Figure 7. These results led us to design our FLWT unit to read all picture data into the internal memory before beginning the transform operation and write the data back to external memory after the transform is completed.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we investigated a hardware acceleration of the DWT with a co-processing unit. The unit supports a new FLWT instruction and was implemented on a reconfigurable platform as an augmentation of a GPP (MIPS). The new instruction was introduced as an ISA extension of the MIPS architecture. For the quantitative analysis, simulations were carried out on a cycle accurate simulator. For benchmarking, we used a publicly available software package [4] and augmented it with new industry approved functionality. We executed the benchmark software both on standard and augmented MIPS architectures and analyzed the results. We worked out the details for 3 different lifting schemes, based on polinomial filters, reversible wavelet transform (Le Gall 5-3 filter) and irreversible wavelet transform (Daubechies 9-7 filter).

TABLE IV

Memory access latencies - different picture dimensions

	Case 1		Case 2	
Picture format	352 × 288	720 × 560	352 × 288	720 × 560
Total number of pixels to be transferred	270 270	1 075 194	101 376	403 200
Total read latency [cycles]	748 920	9 447 987	96 621	655 237
Total write latency [cycles]	270 270	1 075 194	50 688	201 600
Total read/write latency [μsec]	1019	10 523	147	857
HW data transfer limit [μsec]	170	673	170	673
Time for data transfer [μsec]	1 019	10 523	170	857
HW Execution Time [μsec]	3 200	11 720	3 200	11 720
HW execution time [μsec]	**4 219**	**22 243**	**3 370**	**12 577**

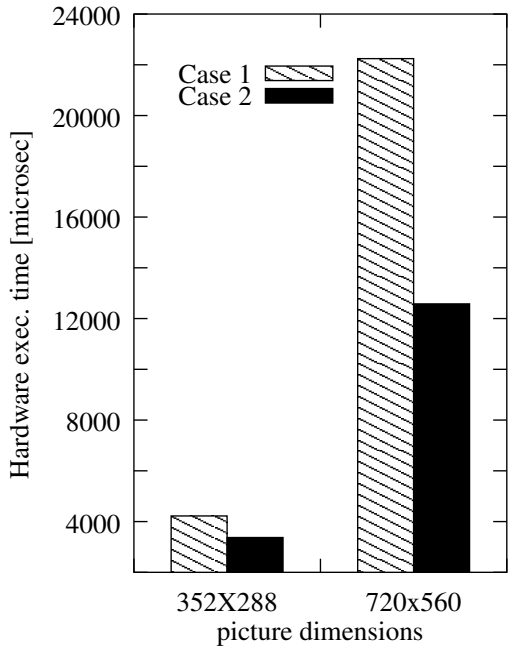

Fig. 7. Comparison of execution time for 1D and 2D transforms

Simulation results indicate that the hardware accelerator can substantially speed up the Lifting Wavelet transform (up to 11 times). Furthermore, the speed gain was found to increase with the picture size and the degree of the filter polynomials. The influence of data memory access latencies on the overal performance of the system was analyzed in two extreme cases of data transfer schemes.

In the future, we intend to implement the investigated FLWT extension into a custom computing machine [8] and to estimate the influence of FPGA reconfiguration time on the overall performance of the system. Processors, augmented with the presented microarchitecture can be efficiently utilized by wavelet based encoding tools and standards like JPEG 2000, MPEG-4, etc.

VII. Acknowledgements

This research is 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] C.M.Brislawn. Classification of nonexpansive symmetric extension transforms for multirate filter banks. *Applied and Comp. Harmonic Analysis*, 3:337–357, 1996.

[2] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. *J. Fourier Anal. Appl.*, 4(3):245–267, 1998.

[3] D.C.Burger and T.M.Austin. The simplescalar tool set, version 2.0. Technical Report CS-TR-1997-1342, University of Wisconsin-Madison, 1997.

[4] G. Fernández, S. Periaswamy, and W. Sweldens. LIFTPACK: A software package for wavelet transforms using lifting. In M. Unser, A. Aldroubi, and A. F. Laine, editors, *Wavelet Applications in Signal and Image Processing IV*, pages 396–408. Proc. SPIE 2825, 1996.

[5] ISO/IEC FCD 15444-1:2000. JPEG 2000 Image Coding System (Final Commettee Draft V1.0), 16 March 2000.

[6] P. Shrestha. MIPS augmented with wavelet transform: Performance analysis. M.Sc. Thesis, Delft University of Technology, July 2002.

[7] W. Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets. *Appl. Comput. Harmon. Anal.*, 3(2):186–200, 1996.

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

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

[10] B. Zafarifar. Micro-codable discrete wavelet transform. M.Sc. Thesis, Delft University of Technology, July 2002.