A Pipelined Architecture for Partitioned DWT Based Lossy Image Compression using FPGA's

Jörg Ritter
Institute for Computer Science
Martin-Luther-University Halle-Wittenberg
06120 Halle, Germany
ritter@informatik.uni-halle.de

Paul Molitor
Institute for Computer Science
Martin-Luther-University Halle-Wittenberg
06120 Halle, Germany
molitor@informatik.uni-halle.de

ABSTRACT

Discrete wavelet transformations (DWT) followed by embedded zerotree encoding is a very efficient technique for image compression [2, 5, 4]. However, the algorithms proposed in literature assume random access to the whole image. This makes the algorithms unsuitable for hardware solutions because of extensive access to external memory. Here, we present an efficient architecture for computing DWT of images, which is based on a partitioned approach for lossy image compression [3]. The architecture achieves its computational power by using pipelining and taking advantage of the flexible memory configurations available in FPGA's.

Categories and Subject Descriptors

I.4.2 [**Image Processing and Computer Vision**]: Compression (Coding)—*Approximate methods*; B.2.1 [**Arithmetic and Logic Structures**]: Design Styles—*pipelining*; B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Algorithms implemented in hardware*

General Terms

Design, Performance, Algorithm

Keywords

pipelining, wavelet transformation, architecture, embedded zero tree coding, FPGA, lossy image compression, field programmable gate arrays, Xilinx

1. INTRODUCTION

Discrete wavelet transformation (DWT) of an image results in a compact multi-scale representation of the image. For detailed information on wavelet transformation we refer to [2]. The wavelet transformed image can be seen as a multi-rooted directed tree. Each node of the tree corresponds to a pixel of the multi-scale representation. The tree is defined in such a way that each node v has either no offspring or four offspring which are a "refinement" of node v (see Figure 1). An excellent choice to encode such wavelet transformed images is applying embedded zero tree (EZT) algorithms which are iterative procedures [5, 4]. In the ith iteration, they start by encoding the roots of the tree, i.e., the tree nodes in which normally most of a wavelet transformed image's energy is concentrated. Then for each pixel of this level, the corresponding subtree is considered. If all the nodes of the subtree are insignificant with respect to the ith threshold T_i, then the offspring of the pixel are not encoded and the subtree is pruned away. If the subtree is not such a zerotree with respect to T_i, the offspring are encoded and the procedure is recursively applied to the offspring. Here, a pixel of the wavelet transformed image is called *insignificant with respect to a threshold* T_i if its magnitude is smaller than T_i. A subtree is called *zerotree with respect to* T_i, if all its nodes are insignificant with respect to T_i. The

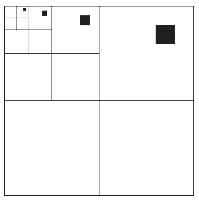

Figure 1: Tree Structure of a wavelet transformed image.

test as to whether a subtree is a zerotree can only be done by considering all the nodes of the subtree until a significant pixel is found. Thus, all pixels of a subtree have to be considered in the worst case. Thus, the EZT algorithms assume that they can efficiently access all the pixels of the image to be encoded. This makes them unsuitable for hardware solutions, in particular for FPGA implementations if the whole image cannot be stored in the internal memory. In [3] we have presented a partitioned approach to perform DWT computations, allowing efficient calculation of the EZT algorithm using programmable hardware. We have discussed a FPGA implementation for lossless image compression.

In this paper, we present an efficient FPGA implementation of the two-dimensional DWT (2D-DWT) for lossy image compression. In particular, the hardware architecture proposed allows compression ratios comparable to software implementations and guarantees, that no block artefacts are introduced. The paper is structured as follows. In Section 2, we give some basic notations. Section 3 and 4 summarize the partitioned approach for lossless and lossy image compression. Section 5 handles a hardware architecture for computation of DWT for lossy image compression, which is the main contribution of our paper. The paper closes with some conclusions.

2. BASIC NOTATIONS

We consider a $n \times n$ image as two-dimensional pixel array $\mathcal{I}$ with n rows and n columns and assume a c-bit greyscale resolution. Furthermore, we assume without loss of generality that the equation $n = 2^k$ holds for some positive integer k.

Partitioning an image $\mathcal{I}$ into $(\frac{n}{q})^2$ quadratic subimages

$$\mathcal{I}^{(0,0)}, \ldots, \mathcal{I}^{(\frac{n}{q}-1, \frac{n}{q}-1)}$$

of size q with $q = 2^r$ for some positive integer $0 \leq r < k$ results in subimages with

$$\mathcal{I}^{(s,t)}[i,j] = \mathcal{I}[s \cdot q + i, t \cdot q + j]$$

for all $0 \leq s, t < \frac{n}{q}$ and $0 \leq i, j < q$.

Furthermore, let $DWT(\mathcal{I})$ denote the wavelet transformed image $\mathcal{I}$. We abbreviate $\mathcal{I}^{(s,t)}[i,j]$ and $\mathcal{I}[l,k]$ with $p_{i,j}^{(s,t)}$ and $p_{l,k}$, respectively. $H(\mathcal{I})$ denotes the first order entropy of image $\mathcal{I}$ and is defined as

$$H(\mathcal{I}) = -\sum_{i,j} P(p_{i,j}) \cdot \log P(p_{i,j})$$

with apriori probabilities

$$P(p_{i,j}) = \frac{\sum_{l,k}(p_{i,j} == p_{l,k})}{n^2}.$$

3. PARTITIONED APPROACH FOR LOSSLESS IMAGE COMPRESSION

The basic idea to save external memory accesses is to partition the image to be encoded into subimages, which can be stored in internal memory. Thus the major task is to design compression algorithms which wavelet transform and encode these subimages without extensive external memory access.

Lossless image compression of partitioned images is quite easy. The subimages $\mathcal{I}^{(s,t)}$ can be wavelet transformed independently of the other subimages. Of course, the resulting transformed image $DWT^q(\mathcal{I})$ which is composed by

$$DWT(\mathcal{I}^{(0,0)}), \ldots, DWT(\mathcal{I}^{(\frac{n}{q}-1, \frac{n}{q}-1)})$$

differs from the conventionally transformed image $DWT(\mathcal{I})$. However, reversibility is ensured. Note that this simple hardware solution works for lossless image compression as no quantization takes place. Figure 2 illustrates this simple approach and demonstrates the potential to parallelize the computation of the wavelet transformation.

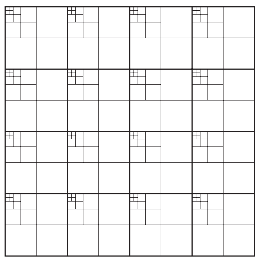

Figure 2: Partitioned wavelet transformation for lossless image compression.

According to the chosen partition size $q \in \{16, 32, 64\}$ computing up to five levels of transformation is possible and reasonable. After having applied the partitioned wavelet transformation, we can apply an EZT algorithm on each partition. Thus, random access is only necessary on the subimage itself which is stored in internal memory. This makes the partitioned approach suitable for a FPGA hardware implementation.

To quantify the effectiveness of the approach, we have measured the first order entropy of both, the conventionally wavelet transformed image $DWT(\mathcal{I})$ and the partitioned wavelet transformed image $DWT^q(\mathcal{I})$ after four levels of transformation. We used the integer-to-integer (5,3)-wavelet [2]. We have applied this particular wavelet because of its very simple filter operations

$$d_i = p_{2i+1} - \left\lfloor \frac{p_{2i} + p_{2i+2}}{2} \right\rfloor \quad (1)$$

$$s_i = p_{2i} + \left\lfloor \frac{d_i + d_{i-1}}{4} \right\rfloor \quad (2)$$

when using the lifting scheme [7]. p_j denotes the j^{th} pixel in the row (column) just under consideration. Note that these two filters can be implemented by additions and shifts. During the transformation steps, modular arithmetic units as proposed by Chao et.al [1] are used. Thus, the transformed image also has c-bit greyscale resolution. Boundary treatment has been done by reflection at image/partition boundaries. The benchmarks used are 8-bit greyscale images of size 512×512, i.e., $c = 8$ and $n = 512$. The size of the subimages has been set to 32.

Table 1 summarizes the results. The first column specifies the image under consideration. The second, third, and fourth column give the first order entropy of the original image $\mathcal{I}$, the conventionally wavelet transformed image $DWT(\mathcal{I})$, and the partitioned wavelet transformed image $DWT^{32}(\mathcal{I})$, respectively. We observe that the first order entropy of a partitioned wavelet transformed image is only slightly higher than the first order entropy of the conventionally wavelet transformed image. This small difference is caused by the reflection at partition boundaries. Thus, the hardware solution proposed allows compression ratios comparable to those guaranteed by wavelet based software solutions.

Table 1: First order entropy $H(\mathcal{I})$

name	$\mathcal{I}$	$DWT(\mathcal{I})$	$DWT^{32}(\mathcal{I})$
airplane	6.70589	4.27120	4.31654
baboon	7.35795	6.16280	6.19905
goldhill	7.47778	4.90245	4.94677
lena	7.44551	4.42375	4.48359
peppers	7.59245	4.73790	4.77402

3.1 FPGA implementation

In order to obtain experimental data on the computation time of the hardware approach just presented, we have implemented the partitioned wavelet transformation on a FPGA prototyping board equipped with a XC4085XLA-HQ240-09 and 2MB external SRAM. After investigation of several implementation alternatives [6], we have decided on an implementation which parallelizes the transformation of one subimage by interlocked computation of the low- and high-pass coefficients. Furthermore, filtering and access to the external SRAM have been shared during the computation of the first level (the whole image is first transfered from the PC to the SRAM of the prototyping card). We achieved clock rates in an order of magnitude of 20 MHz which result in a computation time for wavelet transformation of a 512×512 image of less than 0.02 seconds. Most of the running time is due to huge routing delays caused by the relative large internal memory blocks in the XC4085XLA. We suppose that the hardware solution can be still more speeded up if no restrictions have to be made for the fixed input/output port restrictions of the FPGA board or an FPGA with embedded block RAM is taken as target device. Using such new devices the routing of the address lines will not be the determining factor anymore.

4. LOSSY IMAGE COMPRESSION

If stronger requirements on transmission time or storage space are postulated, higher compression ratios are needed. The only way to meet these requirements is allowing loss of information. This is usually done by quantization after the wavelet transformation had taken place. If we apply the approach presented in the previous section, namely transforming each subimage independently of the neighboring subimages by simply using reflection at the boundaries of the subimage, we obtain block artifacts known from JPEG-compressed images [9].

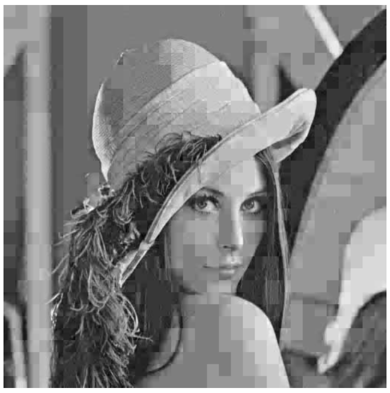

Figure 3: Image transformed by the partitioned DWT algorithm presented in Section 3 and EZT compressed by a factor of 20.

Figure 3 shows a 512×512 8-bit greyscale image $DWT^{32}(\mathcal{I})$ obtained by applying the partitioned DWT algorithms presented in the previous section. The image was compressed by a factor of 20 using a non optimized EZT algorithm without a subsequent arithmetic or huffman coder. Although this compression ratio could be improved while preserving the same image quality, the figure shows that reflection at partition boundaries is no longer feasible because of the block artefacts.

Our objective is that the partitioned compressed image does not differ from the conventionally compressed image. Figure 4 shows the image compressed by a factor 25 if the original image is transformed by a *non-partitioned* DWT algorithm and compressed by a factor 25 by a EZT algorithm. This emphasizes the necessity not to use reflection at partition boundaries, but to work with the original pixels.

Let us examine which pixels have to be considered in order to compute the low- and high-pass coefficients of the (5,3)-wavelet transformation already applied in Section 3. By resolving Equations 1 and 2, we obtain

$$d_i = -\frac{1}{2}p_{2i} + p_{2i+1} - \frac{1}{2}p_{2i+2} \tag{3}$$

$$s_n = -\frac{1}{8}p_{2i-2} + \frac{1}{4}p_{2i-1} + \frac{3}{4}p_{2i} + \frac{1}{4}p_{2i+1} - \frac{1}{8}p_{2i+2}. \tag{4}$$

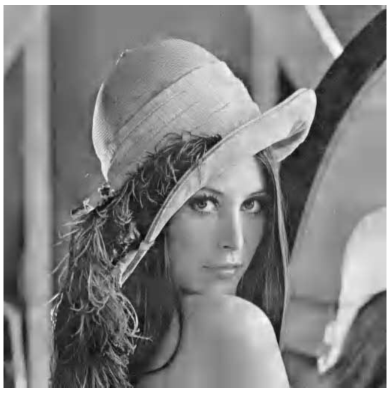

Figure 4: Image transformed by the non-partitioned DWT algorithm and EZT compressed by a factor of 25.

Now, take a look at a pixel row $\mathcal{R}$ of length 16 which consists of pixels $p_0, \ldots, p_{15}$. The pixel row is transformed in a new 16-bit pixel row consisting of 8 low-pass coefficients $s_0, \ldots, s_7$ and of 8 high-pass coefficients $d_0, \ldots, d_7$. These coefficients depend on the pixels $p_{-2}, p_{-1}, p_0, p_1, \ldots, p_{15}, p_{16}$. Note that in the partitioned approach proposed in Section 3 the pixels p_{-2}, p_{-1}, and p_{16} are not available in the internal memory. Thus, in order to compute the coefficients $s_0, \ldots, s_7$ and $d_0, \ldots, d_7$, the transformation has to look out over the right and left boundary of the pixel row $\mathcal{R}$ by one and two pixels, respectively. In the following transformation levels further pixels from outside of the pixel row $\mathcal{R}$ have to be accessed. After four levels of transformation, the wavelet transformation of the 16-bit pixel row $\mathcal{R}$ has to look out over the left and right boundary by 30 and 15 pixels, respectively. Unfortunately, it is too expensive to simply store all these "external" pixels in the internal memory, too. This would result in an internal memory with a capacity which is more than 14 times larger than the internal memory used till now because a subimage of size 61 × 61 has to be stored. We have decided on another approach and will illustrate it in the following section.

5. THE PIPELINED ARCHITECTURE FOR LOSSY IMAGE COMPRESSION

Before we explain the architecture in detail we have to address the given environment. Our experiments are based on a prototyping PCI-card, equipped with a Xilinx-FPGA XC4085XLA-09-HQ240 and external 2MB SRAM as already mentioned in Section 3.1. The

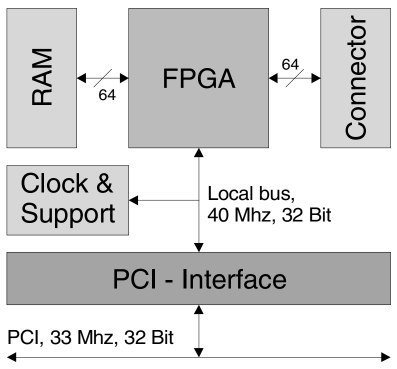

Figure 5: environment of the prototyping card

FPGA is connected to the PCI bus trough a 32-bit wide local bus with a maximum frequency of 40MHz. The external SRAM located on the prototyping card is connected via a 36-bit wide bus and 21-bit address lines and several control signals. Figure 5 illustrates the given environment. The pixels of an image can be transfered to the FPGA in a data stream with 4 pixels at one clock cycle, if we consider at most 8-bit greyscale or 24-bit color images (each color channel is transformed independently of the other). The maximal throughput is therefore given as 152Mbyte per second and already bounded by the maximal throughput of the PCI bus of 132Mbyte/s.

The proposed architecture for the partitioned 2D-DWT mainly consists of two one-dimensional DWT units (1D-DWT) for horizontal and vertical transformations, a control unit realized as a finite state machine, and an internal memory block. For illustration see Figure 6. To process a subimage, all rows are transfered to the

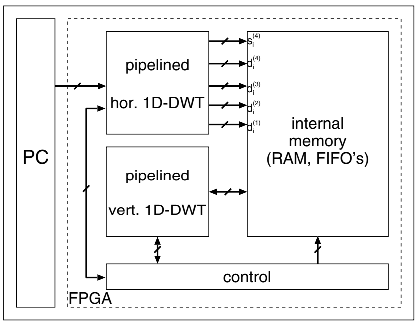

Figure 6: block diagram of the proposed architecture

FPGA via the PCI bus and transformed on the fly in the horizontal 1D-DWT unit using pipelining (see Section 5.1). The coefficients computed in this way are stored in internal memory of different types. The coefficients corresponding to the rows of the subimage itself are stored in single port RAM. The others are shifted into the fifo's (see Section 5.3). Now the vertical transformation levels can take place. This is done by the vertical 1D-DWT unit (see Section 5.2). The control unit coordinates these steps in order to process a whole subimage and is responsible for generating enable signals, address lines, and so on.

At the end, the wavelet transformed subimage is available in the internal RAM. At this point an EZT-algorithm can be applied to the multi-scale-representation of the subimage. Since all necessary boundary information was included in the computation, no block artefacts will be introduced by the following quantization.

The following sections describe the particular units, in detail.

5.1 Horizontal DWT unit

As opposed to the conventional 2D-DWT where horizontal and vertical 1D-DWT alternate, we compute first all four horizontal transformation levels. This allows a pipelined approach because the intermediate results do not have to be transposed. The whole horizontal transformation is done for the 16 rows of the subimage under consideration. In addition, 30 rows of the neighboring subimage in the north and 15 rows of the southern subimage are transformed in the same manner. These additional computations are required by the vertical DWT applied next.

As already mentioned this unit has to take four pixels of a row at each clock cycle and must perform 4 levels of horizontal transformations. Figure 7 illustrates that the unit consists of four pipelined stages, one for each transformation level. The module *4i4o* used in the first and second level will be described in Section 5.1.2. The module *1i2o* is presented in Section 5.1.1, in detail.

The data throughput is mainly dominated by the first stage, since the number of coefficients is down sampled to half after each level of transformation. Therefore the first and second level are performed by a module with higher throughput than the following two levels. The first and second stage outputs two low and two high frequency coefficients at one clock cycle, respectively. After the first stage, the two high frequency coefficients (the ones at even and odd positions) are merged together to output a continuous stream of $d_i^{(1)}$. The two low frequency coefficients are merged together in the same manner. Furthermore they are combined into a four pixels wide bus as input of the second stage. This is done in the module named *2to4* by a simple delay element.

The second stage operates similar to the first one but at only half speed. The computed low frequency coefficients of level two are merged together into a single data stream. This is the input for the third stage. The high frequency coefficients are merged in the same way and are shifted out.

The transformation units of stage three and four (named *1i2o* in Figure 7) take only one coefficient of the previous level as input and alternately outputs a low or a high frequency coefficient at one clock cycle. The horizontal 1D-DWT unit processes a pixel row of length 16 in only 32 input clock cycles including the boundary pixel.

In the following two subsections we specify the modules *1i2o* and *4i4o* used in the different transformation levels.

5.1.1 *The w-bit input 1D-DWT unit (1i2o)*

To implement one level of DWT using the lifting method (see Section 2) the following steps are necessary:

- split the input into coefficient at odd and even positions,
- perform a *predict-step*, that is the operation given in Equation 1,
- perform an *update-step*, that is the operation given in Equation 2.

An efficient realization of the last two steps is given in Figure 8 and Figure 9. The computation is split into the elementary pieces,

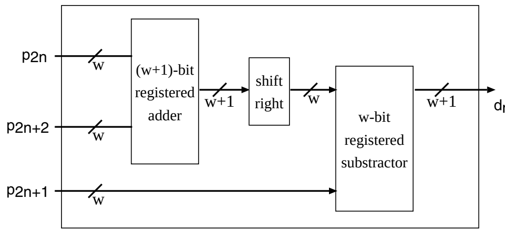

Figure 8: predict unit

which are additions, subtractions and shifts in order to achieve clock rates above 40 MHz. With the given arrangement, the combinational functions between two flip-flops fits into one column of CLB's. Furthermore the registered adder/subtracters generated by Logiblox are configured such that the dedicated carry logic in the XC4000 series is used.

The whole w-bit 1D-DWT unit is constructed accordingly to the lifting scheme [7]. Figure 10 sketches the architecture. The unit consists of two register chains. The registers in the upper chain are enabled at even, the registers in the lower chain at odd clock edges. This splits the input into words at even and odd positions. Now the predict and update steps can be applied straightforward.

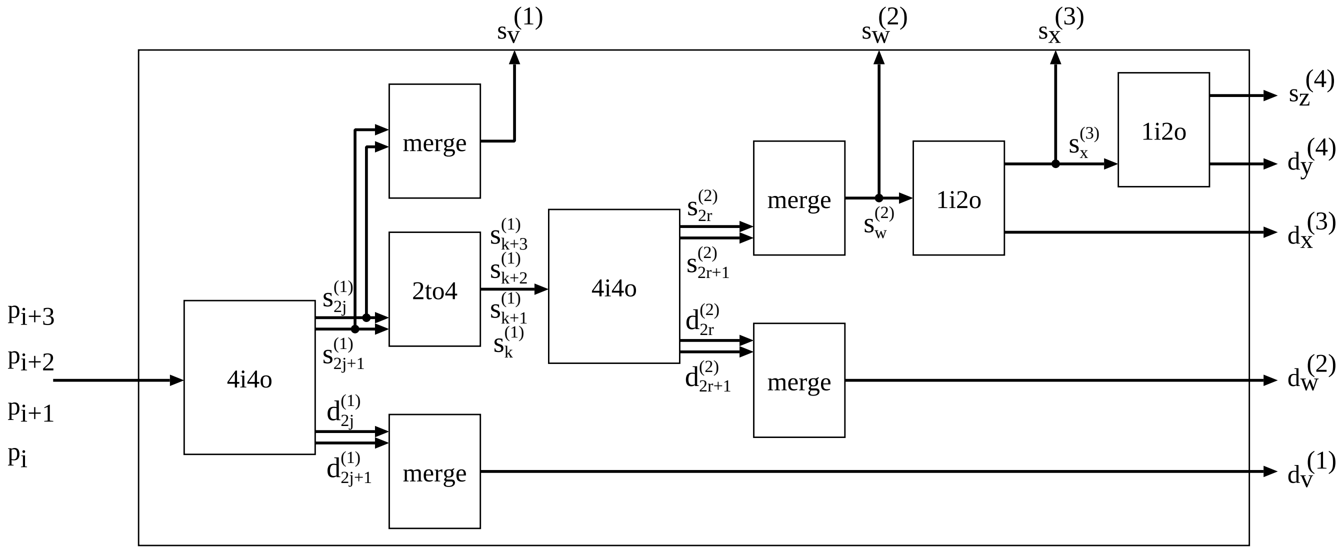

Figure 7: 4 level horizontal 1D-DWT unit

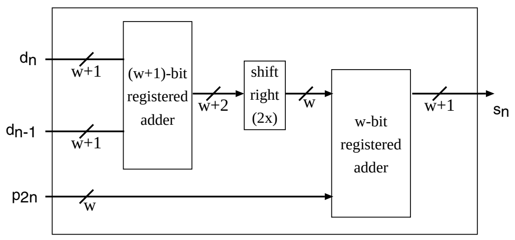

Figure 9: update unit

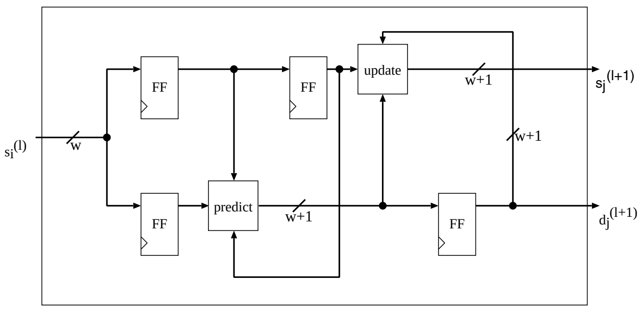

Figure 10: w-bit input 1D-DWT unit

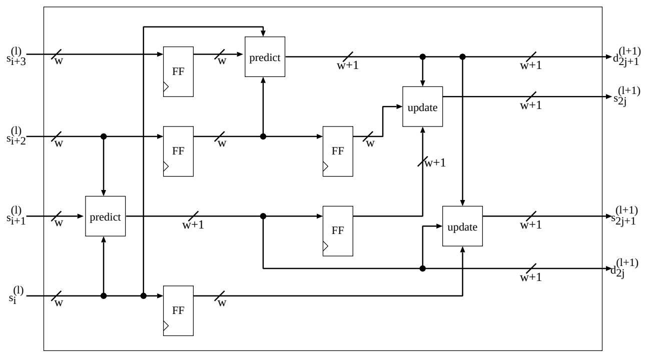

Figure 11: $4w$-bit input 1D-DWT unit

5.1.2 *The $4w$-bit input 1D-DWT unit (4i4o)*

In order to perform a faster transformation, which is needed during the first and second level, the w-bit input 1D-DWT has to be parallelized. One approach is to process four rows in parallel but we have to take into account the growing chip area (factor 4). Another disadvantage is the fact, that we have to split the RAM into four slices, where each slice corresponds to a w-bit input 1D-DWT. This results in additional data and address lines and slow down the access time to the RAM. As a direct consequence of this, the order of the data transfered to the FPGA has to be adapted accordingly.

To avoid the disadvantages just mentioned we have implemented a filter unit as shown in Figure 11, which takes four pixels of the *same* row at a time. Instead of 12 w-bit and 12 $w+1$-bit flip-flops, 4 predict and update units we only need 4 w-bit and 5 $w+1$-bit flip-flops and 2 predict and update components, if the bit width of the input coefficients/pixels is w.

We use this unit for both the first and the second level of the transformation.

5.2 Vertical 1D-DWT unit

The vertical 1D-DWT unit is based only on the w-bit input 1D-DWT component. During the vertical transformation up to five such components can work in parallel, because the memory is already split in vertical slices. The overall structure of this unit (cascaded one level components) is similar to the horizontal one.

5.3 Internal Memory

To perform the partitioned DWT we introduce memory blocks to store the low and high frequency coefficients of the different levels. Our main goal with the partitioned approach in mind was to efficiently perform an EZT-algorithm on an internal saved subimage. As explained in Section 4, pixels belonging to neighboring subimages have to be considered and stored, resulting in larger memory requirements. By applying pipelining, the memory can be reduced by a factor of four. Instead of a RAM block of size 61×61, a block of size 16×61 is sufficient, now. The partition size $q = 16$ was chosen due to the routing capabilities of the Xilinx device. Huge memory blocks slow down the achievable frequency and introduce large routing delays, especially for the address lines to the memory. Because random access is needed on the transformed subimage only, all boundary coefficients are stored in fifo's. Keep in mind, that blocks with different memory depth are necessary, due to the growing bit width after each transformation level.

In order to compute the range of the coefficients of different sub-bands and different levels consider the wavelet transformation in Equation 3 and 4 in terms of FIR filters [8]. The computation of the sub-band coefficients can now be expressed as convolution,

$$d_n = f * x = \sum_k f_{n-k} \cdot x_k$$

$$s_n = g * x = \sum_k g_{n-k} \cdot x_k$$

where f and g are appropriate FIR filters. If we recursively apply the filter g on s_n, we obtain a new FIR filter g^2

$$s_m^{(1)} = g * (g * s_n) = (g * g) * s_n := g^2 * s_n.$$

After determining all filters of each sub-band and each level we can compute the maximum and minimum values of the corresponding coefficients. Table 2 summarizes these values for the one dimensional case.

Table 2: range of coefficients after each level of horizontal transformation

	coefficients			
level i	high frequency		low frequency	
	max	min	max	min
1	255	-255	191	-192
2	160	-160	223	-224
3	136	-136	249	-250
4	103	-103	260	-261

As a consequence our memory modules for the 2D-DWT should have at minimum the bit width given in Figure 12. Note that each

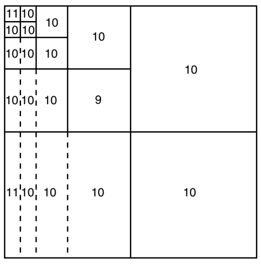

Figure 12: affordable memory bit width

vertical slice is realized by a separate RAM module generated by the Xilinx Logiblox generator.

6. CONCLUSIONS

We have presented a promising FPGA based hardware architecture for DWT in case of lossy and lossless image compression. The FPGA implementation proposed achieves its computational power by using the partitioned approach and taking advantage of pipelining.

Currently, an EZT algorithm based on this 2D-DWT architecture is implemented within the scope of a master's thesis at our institute. In future work, we will investigate the applicability of the approach presented in this paper to video compression, in particular to motion estimation. We suppose that the partitioned approach is suitable for being applied to this area because motion seems to be usually localized through the subimages used. Furthermore, we are looking for methods allowing the partitioned approach to result in higher compaction ratios than the non-partitioned algorithms. One point of departure is to classify the subimages with respect to similarity, to compress only one representative $\mathcal{J}_C$ of each class C, and to represent the remaining subimages $\mathcal{I}$ of class C by the image $\mathcal{I} - \mathcal{J}_C$ which specifies the differences between subimage $\mathcal{I}$ and its representative $\mathcal{J}_C$. Another point of departure is to exploit similarities between the trees of the multi-scale representation.

7. REFERENCES

[1] H. Chao, P. Fisher, and Z. Hua. An approach to integer wavelet transformations for lossless image compression. *Technical Paper, University of North Texas, Denton, TX 76208, U.S.A.*, 1996.

[2] I. Daubechies. *Ten lectures of wavelets*. SIAM, Philadelphia PA, 1992.

[3] J. Ritter and P. Molitor. A partitioned wavelet-based approach for image compression using FPGA's. In *Proceedings of the 2000 Custom Integrated Circuits Conference*, pages 547–550. IEEE, 2000.

[4] A. Said and W. A. Pearlman. A new fast and efficient image codec based on set partitioning in hierarchical trees. In *Trans. Signal Processing*, volume 5, no.9, pages 1303–1310. IEEE, 1996.

[5] J. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. In *Trans. Signal Processing*, volume 11, pages 3115–3162. IEEE, 1993.

[6] S. Sutter. FPGA-architectures for partitioned wavelet transformations on images. Master thesis (in German), Martin-Luther-University Halle-Wittenberg, D-06099 Halle, Germany, 1999.

[7] W. Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets. In *Applied and Computational Harmonic Analysis*, volume 3, no.2, pages 186–200, 1996.

[8] M. Vetterli and J. Kovacevic. *Wavelets and Subband Coding*. Prentice Hall, Inc, New Jersey, 1995.

[9] G. Wallace. The JPEG still picture compression standard. *Comm. ACM*, 34:30–44, 1991.