Single-Loop Approach to 2-D Wavelet Lifting with JPEG 2000 Compatibility

David Barina Petr Musil Martin Musil Pavel Zemcik
Brno University of Technology
Czech Republic
{ibarina,imusilpetr,imusil,zemcik}@fit.vutbr.cz

***Abstract*—A novel approach to 2-D single-loop wavelet lifting with compatibility to JPEG 2000 is presented in this paper. A newly developed 2-D core of CDF 5/3 wavelet filter is presented that, using a new sequence of operations, simplify the design. Moreover, the proposed approach, that uses one pass for 2-D transform, directly produces final output and reduces significantly the need for storing intermediate results into memory. All the proposed structures can be efficiently pipelined in hardware. This paper describes the proposed approach, its implementation in FPGA, cost of such implementation, and brings an experimental evaluation as well as discussion of the features of the approach.**

***Index Terms*—Discrete wavelet transforms, Field programmable gate arrays.**

I. Introduction

Discrete wavelet transform (DWT) is a frequently used signal processing tool with ability to decompose the input signal. In image processing, the transform is usually applied involving the tensor product. In such case, the 2-D transform is implemented as a sequence of 1-D transforms applied on rows and columns. One level of such 1-D transform can be computed utilizing a convolution with two complementary filters. However, on many architectures, a more efficient scheme to calculate the transforms coefficients exists. This scheme is called the lifting scheme and, in contrast to the convolution, it benefits from sharing of intermediate results.

This work is dedicated to an effective implementation of 2-D DWT, namely the one used in JPEG 2000 standard.

JPEG 2000 is an image compression standard intended to supersede previous JPEG standard based on the discrete cosine transform. Unlike the original JPEG, the newer JPEG 2000 is built over the discrete wavelet transform. Such decomposition is usually performed on large image tiles. JPEG 2000 provides lossless as well as lossy compression in a single compression system. In its lossless mode, the standard utilizes a rounded version of the biorthogonal CDF 5/3 wavelet filter usually implemented using the lifting scheme.

Programmable logic devices (FPGAs) are one of the platforms suitable for implementations of wavelet transformation. From memory bandwidth point of view, they cannot be compared to current GPGPU cards. Moreover, the advantage of FPGA implementation is mainly in small embedded devices, such as cameras, which are already based on FPGA and/or have a fixed requirements on real-time processing, dimensions, low resource and power consumption and where the GPGPU or other platforms simply cannot be deployed.

The problem of efficient 2-D DWT implementation was widely studied on various platforms including the FPGA devices. Despite of this fact, so far we have not seen an approach fusing the separated 1-D transforms into a single-loop transform. Accordingly, we address such issue in this paper. As a result, we have got a high performance architecture capable of transform huge 2-D images in a single pass without the need for storing intermediate results into memory. More specifically, we have focused on a single scale decomposition with the integer CDF 5/3 wavelet filter. In contrast to the other works, we are dealing with the seamless transform over the whole image instead small tiles.

In the following Section II, we briefly summarize the related work and explain our motivation. In Section III, two single-loop cores for 2-D wavelet lifting with CDF 5/3 wavelet are proposed. Section IV describes the implementation details related to the JPEG 2000 definition of the transform. Section V presents the results achieved. Finally, Section VI summarizes our work.

II. Related Work

As mentioned above, the 1-D lifting scheme [1] is the traditional scheme for computing the discrete wavelet transform on many architectures. Unlike the convolution, the lifting scheme requires fewer arithmetic operations per one transform coefficient. The DWT calculated through this scheme can be computed in several successive lifting steps. A naive approach of the scheme calculation would directly follow these lifting steps over the entire signal. However, this would be very inefficient for long input signals because the complete signal is accessed many times. Therefore, so called pipelined computation [2] is usually used at this point.

In [3], the authors derived a family of biorthogonal spline wavelets often referred to as CDF wavelets. In the first part of the JPEG 2000 image coding system, two of them are employed while implemented using the lifting factorization from [1]. First one, CDF 9/7 wavelet over real numbers is used in conjunction with the lossy image compression. The second one, the integer-to-integer approximation of CDF 5/3 wavelet is used in the lossless mode. Because this paper is focused on such transform, we review its lifting scheme here. The 1-D input signal $\boldsymbol{x} \in (x_m)_{0 \leq m < M}$ of length M samples

978-1-4673-8621-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SBAC-PADW.2015.10

is split into two disjoint groups comprising s and d coefficients as

$$s_m^{(0)} = x_{2m}, \qquad d_m^{(0)} = x_{2m+1},$$

where $0 \leq 2m+1 < M$. From now on, the bracketed number in superscript indicates the number of elementary lifting operations performed on the given coefficient. This notation is also used in the other papers, e.g. in [1].

Then, the transform defined in JPEG 2000 maps the input pairs $(d_m^{(0)}, s_m^{(0)})$ onto the output ones $(s_{m-1}^{(1)}, d_m^{(1)})$ as

$$d_m^{(1)} = d_m^{(0)} - \left\lfloor \frac{s_{m-1}^{(0)} + s_m^{(0)}}{2} \right\rfloor, \tag{1}$$

$$s_{m-1}^{(1)} = s_{m-1}^{(0)} + \left\lfloor \frac{d_{m-1}^{(1)} + d_m^{(1)} + 2}{4} \right\rfloor. \tag{2}$$

Note, please, the rounding term $+2$ in (2).

Considering the 2-D signals, such as images, the transform can be computed using several strategies. These are typically referred to as the separable transform (row–column), the block-based transform and the line-based transform. We will now briefly review these strategies. Their detailed description can be found, e.g., in [4].

The simplest strategy is to perform the separable transform by subsequent horizontal and vertical passes over the whole input image. This approach requires the use of large off-chip memory blocks to store the intermediate results. Unlike this strategy, the latter two do not require to store the intermediate results into off-chip memory. The block-based and line-based strategies perform the horizontal and vertical filtering onto smaller image fragments. These fragments consist of rectangular areas or small groups of lines in case of the block-based or the line-based strategy, respectively.

In all the previous strategies, the output coefficients are generated in chunks of various sizes. None of them generates the coefficients continuously with granularity corresponding to the essence of 2-D DWT. Note, please, that this elementary granularity of DWT is a quadruple of LL, HL, LH, and HH coefficients.

Let us now take a look at several papers on FPGA implementation of 2-D DWT. In [5], the authors implemented separable transform using the convolution rather than the lifting scheme. However, in contrast with our work, their implementation was able to deal with images of the size of 512×512 samples only, although, as the authors showed, bigger tiles are also possible for the price of much higher BRAM consumption. In [6], the authors proposed a line-based architecture with focus on JPEG 2000. As in previous work, the architecture was able to process the images of the size of 512×512. However, the transform is implemented using the lifting scheme. Another work focused on JPEG 2000 was done in [7]. As in the previous two cases, this implementation can deal with the tiles of the size of 512×512 pixels. Similarly, it is build upon the lifting scheme and process images line by line. Yet another line-based 2-D wavelet transform implementation of JPEG 2000 was proposed in [8]. Again, it is based on the lifting scheme. This time, the implementation deals with 256×256 images. Many other papers can be found. However, none of them address the problem of efficient processing of high resolution, e.g. Full HD or 4K UHD, images.

Recently, in [9], the authors formulated the single-loop core approach to the 2-D wavelet lifting which generates the output coefficients continuously. We will now briefly describe this approach.

The heart of this approach is a 2×2 core of 2-D lifting, which produces a quadruple of coefficients (LL, HL, LH and HH). Compared with previous approaches, the core approach requires two quotable changes in image processing. First, two auxiliary buffers (one for each direction) are introduced in order to exchange the intermediate results between adjacent cores. For image of size of $M \times N$ coefficients, the auxiliary buffers must have a total size of roughly $M+N$ packed words (depending on the lifting scheme). Using the raster image scan, the second (vertical) buffer is not required which reduces the size of auxiliary buffer to roughly M packed words. Considering the FPGAs, it is feasible to hold this buffer in the on-chip memory. The core itself produces resulting coefficients with a lag of F samples in both directions (depending on the lifting scheme). Second, the input image is virtually extended in order to deal with such lag and properly calculate the symmetric (or other) border extensions. This results in a slight increase in the overall computational cost.

However, as the result, every quadruple of input pixels is visited only once while the output coefficients are produced immediately. The particular form of the core is now a subject of our research. For completeness, let us note that the core approach is not necessarily mutually exclusive with the line-based and block-base strategies described above.

III. Cores

In this section, we propose two single-loop 2-D DWT cores suitable for hardware implementation in FPGAs. These cores can be used as standalone computing units or incorporated into the existing block-based or line-based architectures.

Now, let consider the 2-D input signal $\boldsymbol{x} \in (x_{m,n})_{(0,0) \leq (m,n) < (M,N)}$ of size $M \times N$ samples. It is initially split into four disjoint groups (even and odd samples in the horizontal as well as vertical direction) as

$$a_{m,n}^{(0)} = x_{2m,2n}, \qquad v_{m,n}^{(0)} = x_{2m,2n+1},$$
$$h_{m,n}^{(0)} = x_{2m+1,2n}, \qquad d_{m,n}^{(0)} = x_{2m+1,2n+1},$$

where $(0,0) \leq (2m+1, 2n+1) < (M,N)$. At this point, the $\boldsymbol{a}$ samples are referred to as approximation coefficients, the $\boldsymbol{v}$, $\boldsymbol{h}$ and $\boldsymbol{d}$ as vertical, horizontal and diagonal coefficients, respectively. The single-loop approach operates over the input samples $\boldsymbol{x}$ and instantly produces the coefficients $(\boldsymbol{a}, \boldsymbol{v}, \boldsymbol{h}, \boldsymbol{d})$. The core of this approach is a streaming unit which consumes the input signal and produces the resulting coefficients. Such core needs to store the intermediate results in auxiliary buffers between consequent iterations. Due to the pair-wise nature of

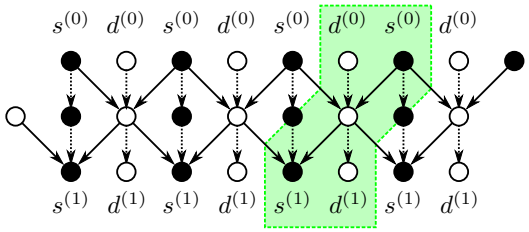

Figure 1. Implementation of CDF 5/3 filter with the highlighted core. Circles represent the results of operations. The notation in accordance with [1].

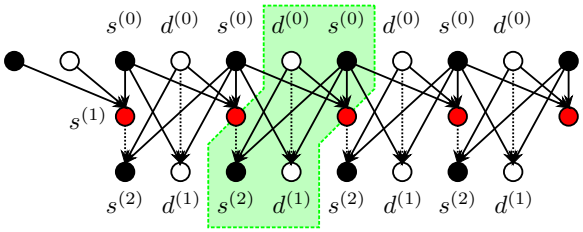

Figure 3. The 1-D implementation of CDF 5/3 filter with the reduced latency. The circles in different colour correspond to $s^{(1)}$ coefficients from (31). The core is highlighted.

the discrete wavelet transform, the smallest possible 1-D core has size of 2 samples. In 2-D case, the minimum possible core size is inherently 2×2 pixels.

In more detail, the core transforms the vector $\boldsymbol{x}$ into $\boldsymbol{y}$. These two vectors are composed from the 2×2 fragment of input signal and some specific intermediate results as

$$\begin{aligned} \mathbf{x} &= \begin{bmatrix} a^{(0)}_{m,n} & h^{(0)}_{m,n} & v^{(0)}_{m,n} & d^{(0)}_{m,n} & \cdots \end{bmatrix}^T \\ \mathbf{y} &= \begin{bmatrix} a^{(.)}_{m-1,n-1} & h^{(.)}_{m,n-1} & v^{(.)}_{m-1,n} & d^{(.)}_{m,n} & \cdots \end{bmatrix}^T \end{aligned} \quad (3)$$

The transform in 2-D is defined by the tensor product of 1-D transforms. It is, therefore, necessary to explain the mapping of 1-D lifting scheme to the 2-D one. The $(a_{m,n}, h_{m,n})$ and $(v_{m,n}, d_{m,n})$ pairs correspond to the (s_m, d_m) pair for the horizontal filtering. Analogously, $(a_{m,n}, v_{m,n})$ and $(h_{m,n}, d_{m,n})$ pairs correspond to the (s_n, d_n) pair in the vertical direction. Please keep this in mind for the rest of this paper.

At the beginning of our work, we adapt the vertical 2×2 core from [9] to the CDF 5/3 transform. As in [1], we define CDF 5/3 lifting scheme using constants $\alpha = -1/2$ and $\beta = 1/4$. The adaptation is fairly straightforward. The resulting core has 4 independent stages suitable for hardware pipelining. These stages are shown in more detail in Figure 2. The first two of them corresponds to horizontal filtering and does not need to access the coefficients in the auxiliary buffer. The latter two corresponds to vertical filtering and needs to exchange the data through the on-chip auxiliary buffer. The core consists of 16 multiply–accumulate (MAC) operations in total. The length of the longest data path in both stages is 2 operations.

The individual stages correspond to predict P^H_α, P^V_α and update U^H_β, U^V_β steps in horizontal and vertical direction, respectively. The core can be then described in matrix notation as

$$\boldsymbol{y} = U^V_\beta P^V_\alpha U^H_\beta P^H_\alpha \boldsymbol{x}. \quad (4)$$

The computation inside the horizontal $U^H_\beta P^H_\alpha$ may be implemented as follows. For better understanding, the 2×1 slice of computation is depicted in Figure 1.

$$d^{(1)}_{m,n} = d^{(0)}_{m,n} + \alpha(v^{(0)}_{m-1,n} + v^{(0)}_{m,n}) \quad (5)$$
$$h^{(1)}_{m,n} = h^{(0)}_{m,n} + \alpha(a^{(0)}_{m-1,n} + a^{(0)}_{m,n}) \quad (6)$$
$$v^{(1)}_{m-1,n} = v^{(0)}_{m-1,n} + \beta(d^{(1)}_{m-1,n} + d^{(1)}_{m,n}) \quad (7)$$
$$a^{(1)}_{m-1,n} = a^{(0)}_{m-1,n} + \beta(h^{(1)}_{m-1,n} + h^{(1)}_{m,n}) \quad (8)$$

Analogously, the vertical $U^V_\beta P^V_\alpha$ follows.

$$v^{(2)}_{m-1,n} = v^{(1)}_{m-1,n} + \alpha(a^{(1)}_{m-1,n-1} + a^{(1)}_{m-1,n}) \quad (9)$$
$$d^{(2)}_{m,n} = d^{(1)}_{m,n} + \alpha(h^{(1)}_{m,n-1} + h^{(1)}_{m,n}) \quad (10)$$
$$a^{(2)}_{m-1,n-1} = a^{(1)}_{m-1,n-1} + \beta(v^{(2)}_{m-1,n-1} + v^{(2)}_{m-1,n}) \quad (11)$$
$$h^{(2)}_{m,n-1} = h^{(1)}_{m,n-1} + \beta(d^{(2)}_{m,n-1} + d^{(2)}_{m,n}) \quad (12)$$

As a second step, we have tried to reduce the depth of the calculation per the output quadruple. The key idea of this is to detach a part of the calculation of the current core which does not depend on the current inputs and merge this part into calculations in the preceding core. During this procedure, we get operations with new scaling factors. Luckily, these are again powers of two as they are composed of the original factors. Finally, we got the core with two stages – one for horizontal and one for vertical filtering. The stages are shown in Figure 4. We have also introduced new intermediate results which replaced part of the original results in the auxiliary buffer. Therefore, the memory consumption of this buffer remains untouched. The newly formed core requires 28 MACs per output quadruple. The length of the critical path in each stage is 2 operations. Thus, the total depth of the entire calculation is smaller (as the number of subsequent stages was halved) than in the original case. In the matrix notation, we have got

$$\boldsymbol{y} = M^V_{\alpha,\beta} M^H_{\alpha,\beta} \boldsymbol{x}. \quad (13)$$

In this case, the horizontal $M^H_{\alpha,\beta}$ is described below. For better understanding, see Figure 3.

$$\begin{aligned} v^{(1)}_{m,n} = v^{(0)}_{m,n} &+ \alpha\beta v^{(0)}_{m-1,n} \\ &+ \beta d^{(0)}_{m,n} + 2\alpha\beta v^{(0)}_{m,n} \end{aligned} \quad (14)$$
$$v^{(2)}_{m-1,n} = v^{(1)}_{m-1,n} + \beta d^{(0)}_{m,n} + \alpha\beta v^{(0)}_{m,n} \quad (15)$$
$$d^{(1)}_{m,n} = d^{(0)}_{m,n} + \alpha v^{(0)}_{m-1,n} + \alpha v^{(0)}_{m,n} \quad (16)$$
$$\begin{aligned} a^{(1)}_{m,n} = a^{(0)}_{m,n} &+ \alpha\beta a^{(0)}_{m-1,n} \\ &+ \beta h^{(0)}_{m,n} + 2\alpha\beta a^{(0)}_{m,n} \end{aligned} \quad (17)$$
$$a^{(2)}_{m-1,n} = a^{(1)}_{m-1,n} + \beta h^{(0)}_{m,n} + \alpha\beta a^{(0)}_{m,n} \quad (18)$$
$$h^{(1)}_{m,n} = h^{(0)}_{m,n} + \alpha a^{(0)}_{m-1,n} + \alpha a^{(0)}_{m,n} \quad (19)$$

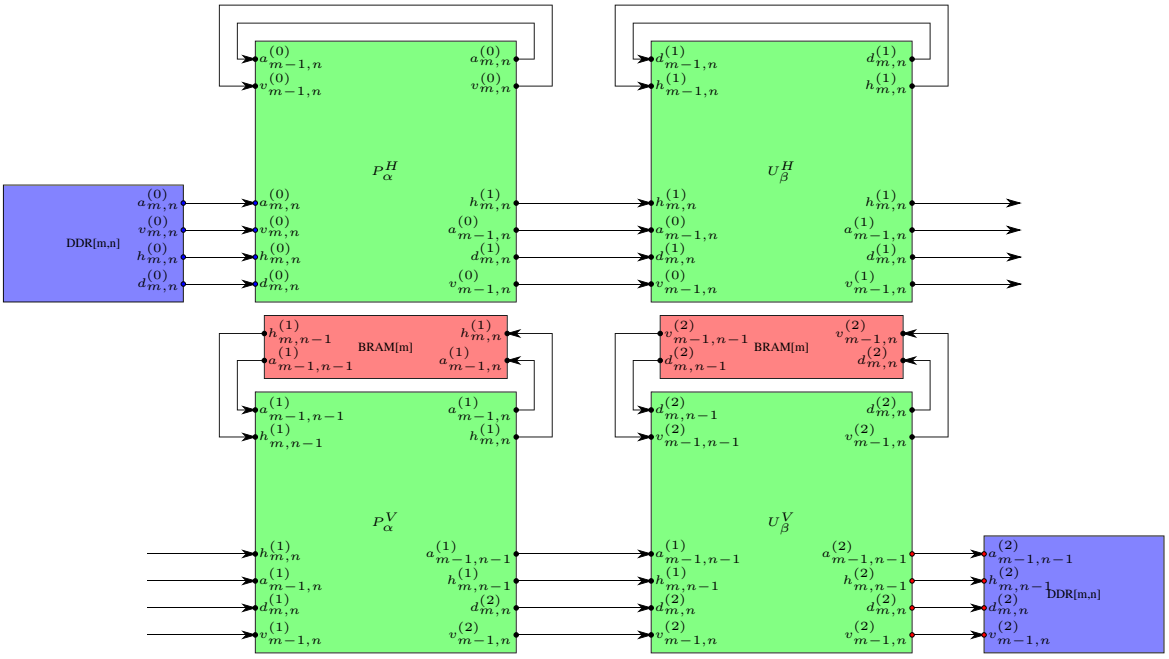

Figure 2. The four-stage separable core with 16 additions only. The arrows pointing to the right are linked to the arrows coming in from the left.

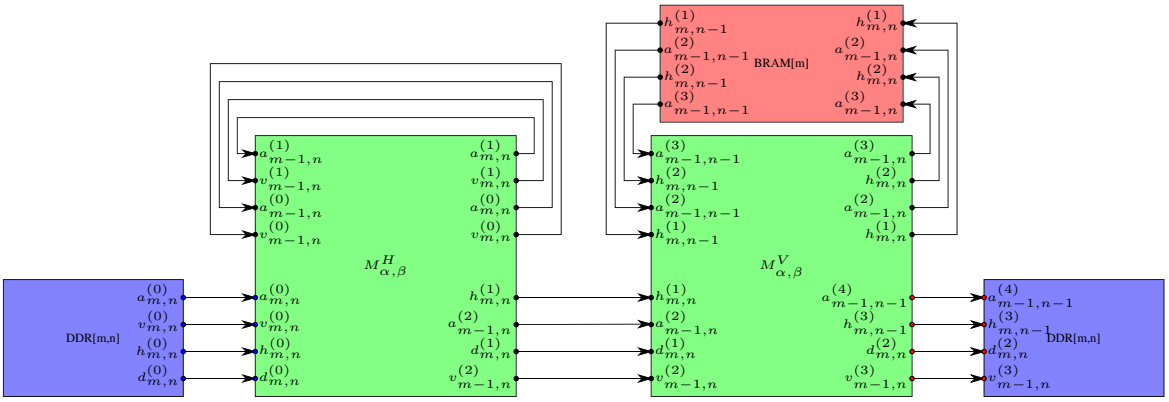

Figure 4. The two-stage separable core with 28 additions.

Analogously, the vertical $M_{\alpha,\beta}^V$ is defined in the following relations.

$$a_{m-1,n}^{(3)} = a_{m-1,n}^{(2)} + \alpha\beta a_{m-1,n-1}^{(2)} + \beta v_{m-1,n}^{(2)} + 2\alpha\beta a_{m-1,n}^{(2)} \quad (20)$$

$$a_{m-1,n-1}^{(4)} = a_{m-1,n-1}^{(3)} + \beta v_{m-1,n}^{(2)} + \alpha\beta a_{m-1,n}^{(2)} \quad (21)$$

$$v_{m-1,n}^{(3)} = v_{m-1,n}^{(2)} + \alpha a_{m-1,n-1}^{(2)} + \alpha a_{m-1,n}^{(2)} \quad (22)$$

$$h_{m,n}^{(2)} = h_{m,n}^{(1)} + \alpha\beta h_{m,n-1}^{(1)} + \beta d_{m,n}^{(1)} + 2\alpha\beta h_{m,n}^{(1)} \quad (23)$$

$$h_{m,n-1}^{(3)} = h_{m,n-1}^{(2)} + \beta d_{m,n}^{(1)} + \alpha\beta h_{m,n}^{(1)} \quad (24)$$

$$d_{m,n}^{(2)} = d_{m,n}^{(1)} + \alpha h_{m,n-1}^{(1)} + \alpha h_{m,n}^{(1)} \quad (25)$$

IV. Implementation

In the previous part, we have proposed two single-loop cores suitable for CDF 5/3 wavelet. Here, we describe the implementation in full detail including correct JPEG 2000 rounding.

The simplest four-stage separable core implements the equations (1) and (2). We have rewritten these equations in order to utilize the input carry bit of the adders.

$$d_m^{(1)} = d_m^{(0)} - \left\lfloor \frac{s_{m-1}^{(0)} + s_m^{(0)}}{2} \right\rfloor \quad (26)$$

$$s_{m-1}^{(1)} = \left\lfloor \frac{4s_{m-1}^{(0)} + d_{m-1}^{(1)} + d_m^{(1)} + 1 + 1}{4} \right\rfloor \quad (27)$$

We have decided to implement the transform using 16-bit

signed integers. Such width is sufficient for few levels of DWT as required by JPEG 2000 standard. The stages reflects the horizontal and vertical predicts and updates, thus we have 4 stages. The auxiliary coefficient between the horizontal stages are passed in registers. In contrary, the vertical intermediate results are exchanged through the Block RAM (BRAM). As all the factors in CDF 5/3 are powers of two, the complete transform consists of 16 additions per output quadruple only.

The implementation of the two-stage core is more sophisticated. At the beginning, we will focus on 1-D transform. Considering the core approach, the biggest problem with the lossless DWT as required by JPEG 2000 is the correct rounding of the s_{m-1} coefficient. Now, the s_{m-1} coefficient is computed in two stages. Consequently, we had to distribute a rounding flag (or a rounding bit) from the first stage of the core to the second one. As the s_{m-1} is computed in two stages, we introduced auxiliary $s^{(1)}_{m-1}$ coefficient which is passed between the two stages. The final coefficients are then denoted $(s^{(2)}_{m-1}, d^{(1)}_m)$.

To explain the two-stage rounding we must first establish the following identity. Let $\mathbb{Z}$ denote the set of integers, $2\mathbb{Z}+1$ denote the set of odd integers, and $p, q \in \mathbb{Z}$. One can perform the following expansion

$$\left\lfloor \frac{p+q}{2} \right\rfloor = \lfloor p/2 \rfloor + \lfloor q/2 \rfloor + C_2(p,q), \tag{28}$$

where C_2 is a correction term is defined as

$$C_2(p,q) = \begin{cases} 1 : p \in 2\mathbb{Z}+1 \wedge q \in 2\mathbb{Z}+1 \\ 0 \end{cases}. \tag{29}$$

Using this identify, we can rewrite the original 1-D transform as follows.

$$d^{(1)}_m = d^{(0)}_m - \left\lfloor \frac{s^{(0)}_{m-1} + s^{(0)}_m}{2} \right\rfloor \tag{30}$$

$$s^{(1)}_m = 4s^{(0)}_m - 2\lfloor s^{(0)}_m/2 \rfloor + d^{(0)}_m - \lfloor s^{(0)}_{m-1}/2 \rfloor \tag{31}$$

$$b_m = 1 - C_2(s^{(0)}_{m-1}, s^{(0)}_m) \tag{32}$$

$$s^{(2)}_{m-1} = \left\lfloor \frac{s^{(1)}_{m-1} + d^{(0)}_m - \lfloor s^{(0)}_m/2 \rfloor + b_{m-1} + b_m}{4} \right\rfloor \tag{33}$$

As in the previous case, the horizontal intermediate values are passed in registers and the vertical ones in BRAM. The core requires 28 additions per output quadruple. Note that the calculation of b_m is implemented in a single NAND gate.

V. Evaluation

We have implemented both of the cores described above. This section brings an experimental evaluation as well as discussion of the features of the approach.

The wavelet engine was experimentally synthesized in a Xilinx Zynq XC7Z045 FPGA and evaluated on the Xilinx ZC706[1] board (with DDR3 at 1066 MHz). The engine was synthesized for several image resolutions (as seen in Table I) that merely differ in the BRAM size only to allow comparison with other papers and also to show that the core is able to process Full HD video (1080p, 60 Hz) faster than in real-time. As it could be seen, the core is ready to process image resolution of up to 4K UHD with the outlook to even higher resolutions without need of any fundamental changes. The computational engine has standardized AXI-Stream interfaces. The input expects streamed video frames in the predefined resolution, the output stream is generating interlaced coefficients of wavelet transform that can be easily split into four separate data streams for further multi-scale decomposition. We would like especially to highlight the ability to process the video stream without the need to use external memory for intermediate results. The design includes mirroring on the image edges which is not performed by wavelet core itself, but by the engine, which encapsulates the core. The engine itself then represents an independent block, which can be used in more complex system or which can be easily duplicated and chained to perform more levels of wavelet transform of one image.

[1] http://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html

	small tile	Full HD	4K UHD
resolution	512×512	1920×1080	3840×2160
theoretical framerate	3698	477	120
framerate with DRAM	2670	338	84

Table I
THE FRAMERATES ACHIEVED FOR VARIOUS IMAGE RESOLUTIONS.

Both of the implemented cores are fully pipelined. First of the cores has latency of 2 clock cycles and it represents the four-stages separable wavelet transform from Figure 2. The second core has latency of 4 clock cycles and it represents the two-stages separable wavelet transform from Figure 4.

	small tile	Full HD	4K UHD
FF	328	332	338
LUT	239	247	257
BRAM	1	2	4

Table III
THE RESOURCES CONSUMED BY THE 4 CLOCK LATENCY CORE.

	small tile	Full HD	4K UHD
FF	280	282	284
LUT	418	440	426
BRAM	1	2	4

Table IV
THE RESOURCES CONSUMED BY THE 2 CLOCK LATENCY CORE.

The wavelet core itself requires just a very small fraction of ZC706 resources, as shown in Table III and in Table IV. As it could be predicted, first solution with 2-stages pipeline leads in higher LUT requirements and less demand for Flip-Flops. At the opposite the 4-stage pipeline consumes more Flip-Flops and smaller amount of LUTs. The requirements for whole wavelet engine are summarized in Table V; besides the core itself, it shows resource demands for engine performing row and column edge mirroring. According to JPEG 2000 standard, the four image lines and columns have to be mirrored, this consumes the extra four BRAMs in case of Full HD resolution.

architecture	device	BRAM [bits]	clocks/pel	time [ms]	FF/reg.
Dillen *et al.* (2003) [8]	VirtexE1000-8	50K	0.50	1.20	2542
Descampe *et al.* (2004) [7]	Virtex-II XC2V6000	N/A	0.60	1.75	N/A
Seo *et al.* (2007) [6]	Altera Stratix	128K	2.64	6.02	N/A
Zhang *et al.* (2012) [5]	Virtex-II Pro XC2VP30-7	6×18K	0.50	0.97	1059
proposed	Zynq XC7Z045	1×36K	**0.26**	**0.27**	280

Table II
COMPARISON OF VARIOUS FPGA IMPLEMENTATION FOR TILES OF SIZE 512×512. THE PROCESSING TIME AND CLOCKS PER PIXEL WERE PROJECTED TO THE UNIFORM IMAGE SIZE. THE BEST RESULTS ARE IN BOLD.

core	FF		LUT		BRAM	
latency 4	441	(0.1 %)	**399**	(0.18 %)	6	(1.1 %)
latency 2	**391**	(< 0.1 %)	592	(0.27 %)	6	(1.1 %)

Table V
RESOURCES CONSUMED FOR FULL HD RESOLUTION ON ZC706 BOARD.

The overall performance of wavelet engine is summarized by Table I. It incorporates the theoretical and real performance of the engine with relation to image resolution. Both of the wavelet cores have the same throughput – they differ just in output latency. The core is able to process 4 input samples in one clock cycle, producing 4 output samples and the important fact also is that each of the samples needs to be fetched from an external memory and stored into the external memory just once as the design is single pass streaming 2-D DWT unit. The computation has to also be performed on the mirrored image edges that are enlarged by 4 pixels in each direction. The theoretical performance was calculated for maximum speed 250 MHz with respect to edge mirroring, assuming ideal situation that input data are always available. The practical performance was measured on hardware Xilinx ZC706. There it could be observed that the RAM throughput is essential for the overall performance.

The overall comparison with the selected architectures is shown in Table II. We have made all the used source codes (including VHDL as well as the reference codes in C language) freely available.[2]

VI. CONCLUSION

We proposed two cores that allow computing the 2-D discrete wavelet transform on a streaming basis. Moreover, we have implemented two high performance engines as standalone processing units. The cores can also be incorporated into some existing applications, namely those using the block-based or the line-based scheduling. Our implementations are built using the integer-to-integer CDF 5/3 wavelet filter with JPEG 2000 standard compatibility. The cores allow for seamless transform calculated over large resolution images instead the small tiles.

We have developed and evaluated the engine with both of the wavelet cores on Xilinx ZC706 board. The engine is small and independent block capable of stream video processing; thus it can be easily duplicated and chained to perform more levels of wavelet transform on one image. Our stream processing approach very efficient from the data throughput point of view. It does not require any external RAM memory, all the necessary intermediate results are effectively stored in local BRAMs. The computation pipeline is simple and allows for high operational frequency (up to 250 MHz) with very low resource consumption. Moreover, the proposed architecture reduces control logic complexity as the horizontal and vertical passes are fused into the single pass. All these features makes this engine useful as a part of pipeline in real-time image processing applications, for example tone-mapping, JPEG 2000 compression, etc.

Future research includes experimenting with multi-scale transforms, different wavelet bases as well as exploitation of the transforms in applications.

[2]http://www.fit.vutbr.cz/research/prod/?id=433

ACKNOWLEDGEMENT

This work has been supported by the IT4Innovations Centre of Excellence (no. CZ.1.05/1.1.00/02.0070) and the TACR Competence Centres project V3C – Visual Computing Competence Center (no. TE01020415).

REFERENCES

[1] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *Journal of Fourier Analysis and Applications*, vol. 4, no. 3, pp. 247–269, 1998. DOI:10.1007/BF02476026

[2] C. Chrysafis and A. Ortega, "Minimum memory implementations of the lifting scheme," in *Proceedings of SPIE, Wavelet Applications in Signal and Image Processing VIII*, ser. SPIE, vol. 4119, 2000, pp. 313–324. DOI:10.1117/12.408615

[3] A. Cohen, I. Daubechies, and J.-C. Feauveau, "Biorthogonal bases of compactly supported wavelets," *Communications on Pure and Applied Mathematics*, vol. 45, no. 5, pp. 485–560, 1992. DOI:10.1002/cpa.3160450502

[4] M. E. Angelopoulou, K. Masselos, P. Y. K. Cheung, and Y. Andreopoulos, "Implementation and comparison of the 5/3 lifting 2D discrete wavelet transform computation schedules on FPGAs," *Journal of Signal Processing Systems*, vol. 51, no. 1, pp. 3–21, 2008. DOI:10.1007/s11265-007-0139-5

[5] C. Zhang, C. Wang, and M. O. Ahmad, "A pipeline VLSI architecture for fast computation of the 2-D discrete wavelet transform," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 8, pp. 1775–1785, Aug. 2012. DOI:10.1109/TCSI.2011.2180432

[6] Y.-H. Seo and D.-W. Kim, "VLSI architecture of line-based lifting wavelet transform for motion JPEG2000," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 2, pp. 431–440, Feb. 2007. DOI:10.1109/JSSC.2006.889368

[7] A. Descampe, F. Devaux, G. Rouvroy, B. Macq, and J.-D. Legat, "An efficient FPGA implementation of a flexible JPEG2000 decoder for digital cinema," in *12th European Signal Processing Conference (EUSIPCO)*, 2004.

[8] G. Dillen, B. Georis, J.-D. Legat, and O. Cantineau, "Combined line-based architecture for the 5-3 and 9-7 wavelet transform of JPEG2000," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 9, pp. 944–950, Sep. 2003. DOI:10.1109/TCSVT.2003.816518

[9] D. Barina and P. Zemcik, "Vectorization and parallelization of 2-D wavelet lifting," *Journal of Real-Time Image Processing*, 2015. DOI:10.1007/s11554-015-0486-6