An Lifting-Based Discrete Wavelet Transform and Discrete Wavelet Packet Processor with Support for Higher Order Wavelet Filters

Andre Guntoro and Manfred Glesner

Department of Electrical Engineering and Information Technology
Institute of Microelectronic Systems
Technische Universität Darmstadt
Karlstr. 15, 64283 Darmstadt, Germany
{guntoro,glesner}@mes.tu-darmstadt.de

Summary. The major challenge in the wavelet transforms is that there exist different classes of wavelet filters for different kinds of applications. In this chapter, we propose a generalized lifting-based wavelet processor that can perform various forward and inverse Discrete Wavelet Transforms (DWTs) and Discrete Wavelet Packets (DWPs) that also supports higher order wavelet filters. Our architecture is based on Processing Elements (PEs) which can perform either prediction or update on a continuous data stream in every two clock cycles. We also consider the normalization step which takes place at the end of the forward DWT/DWP or at the beginning of the inverse DWT/DWP. Because different applications require different number of samples for the transforms, we propose a flexible memory size that can be implemented in the design. To cope with different wavelet filters, we feature a multi-context configuration to select among various forward and inverse DWTs/DWPs. For the 16-bit implementation, the estimated area of the proposed wavelet processor with 8 PEs configuration and 2×2×512 words memory in a 0.18-μm technology is 2.5 mm square and the estimated operating frequency is 319 MHz.

1 Introduction

For the last two decades the wavelet theory has been studied extensively [4, 7, 11, 17, 19] to answer the demand for better and more appropriate functions to represent signals than the ones offered by the Fourier analysis. Contrary to the Fourier analysis, which decomposes signals into sine and cosine functions, wavelets study each component of the signal on different resolutions and scales. In analogy, if we observe the signal with a large *window*, we will get a coarse feature of the signal, and if we observe the signal with a small *window*, we will extract the details of the signal.

One of the most attractive features that wavelet transforms provide is their capability to analyze the signals which contain sharp spikes and discontinu-

ities. The better energy compacting support the wavelet transforms offer and also the localizing feature [5] of the signal in both time and frequency domains these transforms support have made wavelet outperforms the Fourier transform in signal processing and has made itself into the new standard of JPEG2000 [9, 15].

Along with recent trends and research focuses in applying wavelets in image processing, the application of wavelets is essentially not only limited to this area. The benefits of wavelets have been studied by many scientists from different fields such as mathematics, physics, and electrical engineering. In the field of electrical engineering wavelets have been known with the name multirate signal processing. Due to numerous interchanging fields, wavelets have been used in many applications such as image compression, feature detection, seismic geology, human vision, etc.

Contrary to the Fourier transform, which uses one basis function (and its inverse) to transform between domains, there are different classes of wavelet kernels which can be applied on the signal depending on the application. Because different applications require different treatments, researchers have tried to cope with their own issues and implemented only a subset of wavelets which are suitable for their own needs such as ones that can be found in image compression [6, 10, 15, 22] and speech processing [1, 8, 14, 16]. The power of wavelet tools is then limited due to these approaches.

In this chapter we propose a novel architecture to compute forward and inverse transforms of numerous DWTs (Discrete Wavelet Transforms) and also DWPs (Discrete Wavelet Packets) based on their lifting scheme representations. Most lifting-based wavelet processors are dedicated to compute wavelet filters which are used only in JPEG2000 image compression where the wavelet coefficients can be represented as integers such as Andra in [2] which required two adders, one multiplier, and one shifter on each row and column processor to compute (5,3) and (9,7) filters with the prerequisite that prediction or update constants of the actual and the delayed samples are equal (i.e. $c(1+z^{-1})$). Barua in [3] described the similar architecture for FPGAs that optimizes the internal memory usage. Dillen in [13] detailed the combined architecture of (5,3) and (9,7) filters for JPEG2000. Another example is from Martina, which encompassed multiple MAC structure with recursive architecture in [18].

Our new proposed architecture takes into account that each lifting step representation of an arbitrary wavelet filter may have two different update constants and the Laurent polynomial may have higher order factors (i.e. $c_1 z^{-p} + c_2 z^{-q}$), which are common in various classes of wavelet filters such as Symlet and Coiflet wavelet filters. Additionally, the proposed architecture also considers the normalization step which takes place at the end of the forward DWT/DWP or at the beginning of the inverse DWT/DWP for the applications that require to conserve the energy during the transform. In order to be flexible, the proposed architecture provides a multi-context configuration to choose between various forward and inverse DWTs/DWPs. Because wavelet transforms work with large number of samples, the proposed architecture can

be configured to have an arbitrary memory size (i.e. the powers of two) to cope with the application demands.

The rest of the chapter is organized as follows. Section 2.1 describes the second generation of wavelets and the concepts regarding wavelet transforms and wavelet packets. The proposed architecture, including the processing element, the MAC-unit, the configuration and the context switch, the memory, the controller, are explained in Section 3. Section 4 discusses the performance of the proposed architecture and finally Section 5 concludes the contribution.

2 Backgrounds

2.1 Lifting Scheme

Contrary to the filter approach, which separates the signal into low and high frequency parts and performs the decimation on both signals afterwards, the second generation of wavelets reduces the computation by performing the decimation in advance. The second generation of wavelets, more popular under the name of lifting scheme, was introduced by Sweldens [21]. The basic principle of lifting scheme is to factorize the wavelet filter into alternating upper and lower triangular 2×2 matrix.

Let $H(z)$ and $G(z)$ be a pair of low-pass and high-pass wavelet filters:

$$H(z) = \sum_{n=k_l}^{k_h} h_n z^{-n} \tag{1}$$

$$G(z) = \sum_{n=k_l}^{k_h} g_n z^{-n} \tag{2}$$

where h_n and g_n are the corresponding filter coefficients. $N = |k_h - k_l| + 1$ is the filter length and the corresponding Laurent polynomial degree is given by $h = N - 1$. By splitting the filter coefficients into even and odd parts, the filters can be rewritten as:

$$H(z) = H_e(z^2) + z^{-1} H_o(z^2) \tag{3}$$

$$G(z) = G_e(z^2) + z^{-1} G_o(z^2) \tag{4}$$

and the corresponding polyphase representation is:

$$P(z) = \begin{bmatrix} H_e(z)\ G_e(z) \\ H_o(z)\ G_o(z) \end{bmatrix} \tag{5}$$

Daubechies and Sweldens in [12, 21] have shown that the polyphase representation can always be factored into lifting steps by using the Euclidean algorithm to find the greatest common divisors. Thus the polyphase representation becomes:

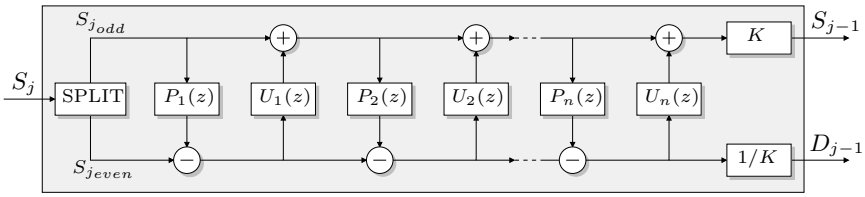

Fig. 1. Forward lifting steps.

$$P(z) = \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \prod_{i=n}^{1} \begin{bmatrix} 1 & a_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ b_i(z) & 1 \end{bmatrix} \tag{6}$$

where $a_i(z)$ and $b_i(z)$ are the Laurent polynomials and K is the normalization factor.

Fig. 1 shows the arrangement of the lifting scheme representation. The Laurent polynomials $b_i(z)$ and $a_i(z)$ are expressed as predictor $P_i(z)$ and updater $U_i(z)$. The signal S_j is split into even and odd parts. Prediction and update steps occur alternately. The predictor $P_i(z)$ predicts the odd part from the even part. The difference between the odd part and the predicted part is computed and used by the updater $U_i(z)$ to update the even part. At the end, the low-pass and the high-pass signals are normalized with a factor of K and $1/K$ respectively.

By factoring the wavelet filters into lifting steps, it is expected that the computation performed on each stage (either it is a prediction or an update) will be much less complex. As an example, the famous Daub-4 wavelet filter with the low-pass filter response:

$$H(z) = \frac{1+\sqrt{3}}{4\sqrt{2}} + \frac{3+\sqrt{3}}{4\sqrt{2}}z^{-1} + \frac{3-\sqrt{3}}{4\sqrt{2}}z^{-2} + \frac{1-\sqrt{3}}{4\sqrt{2}}z^{-3} \tag{7}$$

can be factored into lifting steps:

$$P(z) = \begin{bmatrix} \frac{\sqrt{3}-1}{\sqrt{2}} & 0 \\ 0 & \frac{\sqrt{3}+1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & -z \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\frac{\sqrt{3}}{4} + \frac{2-\sqrt{3}}{4}z^{-1} & 1 \end{bmatrix} \begin{bmatrix} 1 & \sqrt{3} \\ 0 & 1 \end{bmatrix} \tag{8}$$

Since the finding of the greatest common divisors is not necessarily unique, the result of the Laurent polynomials may also differ. The Daub-6 and the popular (5,3) and (9,7) wavelet filters can be factored into lifting steps with maximum degree of ± 1 [12] whereas Symlet-6 and Coiflet-2 (the lifting computations are not detailed here due to page limitation) may have two update/prediction terms and also $z^{\pm 5}$ factor on its Laurent polynomials.

2.2 Wavelet Transform and Wavelet Packet

Wavelet transform is a multi-resolution signal analysis. In the traditional wavelet transforms, only the low-pass signal is used on the next transformation level to generate a multi-resolution representation of the corresponding

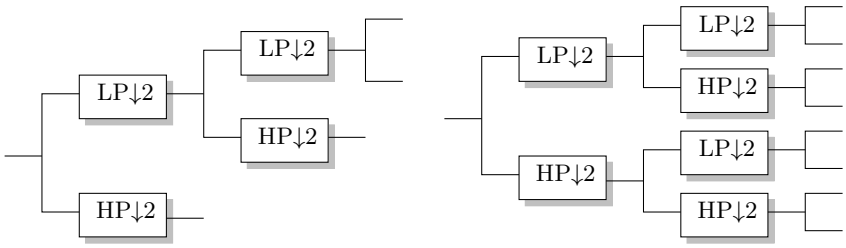

Fig. 2. Two different transformations.

signal. In wavelet packets, both low-pass and high-pass signals are analyzed, resulting equally spaced frequency bands. Fig. 2 depicts both schemes. Note that the illustration uses wavelet transforms based on filter-approach instead of lifting-scheme in order to ease understanding the concept for both schemes. LP and HP correspond to low-pass and high-pass filter pair and ↓2 corresponds to down-sampling by two. It is obvious that DWT will require less computation time compared to DWP, because at each level, the number of samples is decreased by two. Also, the controller that controls the processor to perform DWTs and their inverses is straightforward, while the controller to perform DWPs and their inverses is more complicated due to the fact that the number of frequency bands that need to be processed increases two fold at each transform. As an example, performing four levels wavelet packet on a signal leads to 16 frequency bands whereas performing four levels wavelet transform generates 5 frequency bands.

Not only the challenges on the controller, the major issue in DWP is that the resulting HP signals are much smaller than the LP parts in normal circumstances. Thus performing multi-level DWP using integer arithmetics would make these HP signals go to zero, which lead to lower achievable SNR values, if it is not carefully performed.

3 Proposed Architecture

The lifting-based forward DWT/DWP splits the signal into even and odd parts at the first stage. The split signals are processed by an alternating series of predictors and updaters (on some wavelet filters, an updater may come before a predictor). On the final stage, the multiplication with the normalization factor takes place in order to conserve the energy. The inverse DWT/DWP performs exactly everything backwards. It starts with the multiplication with normalization factor, continues with a series of updaters and predictors, and finishes with the merging of the outputs.

As a predictor and an updater perform a similar computation, the hardware architecture for both functions is exactly the same. Taking this into ac-

count, we propose a novel wavelet processor which is based on M processing elements to cope with M lifting steps. Due to the nature of the lifting scheme, wavelet filters that have longer lifting scheme representations can easily be broken down into smaller lifting steps that the processor can compute (i.e. M lifting steps each). Which means that the processor that implements M processing elements is not limited to perform the transform up to M lifting steps only.

The core behind our proposed architecture is the processing element (PE), which performs the prediction or the update. To maximize the performance, the PE utilizes the parallelism by using a pipeline mechanism to guarantee the outputs to be available in every clock cycle (actually every two clock cycles as detailed later). As the lifting scheme breaks a wavelet filter into smaller predictions and updates, the resulting predictor and updater can be limited to have a maximum Laurent polynomial degree of one. Nevertheless, the predictor or the updater of higher order wavelet filters may have the higher factors as well. Without loss of generality, we can formulate the predictor or the updater polynomial as:

$$l(z) = c_1 z^{-p} + c_2 z^{-q} \tag{9}$$

with c_1 and c_2 as the polynomial constants and $|p-q| \leq N$. This implies that on each stage (either as a predictor or an updater), the PE would perform two multiplications and two additions. As an example, the first predict and update steps of Daub-4 can be written as:

$$\begin{bmatrix} s' \\ d \end{bmatrix} = \begin{bmatrix} 1 & \sqrt{3} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ d \end{bmatrix} = \begin{bmatrix} s + d \cdot \sqrt{3} \\ d \end{bmatrix} \tag{10}$$

$$\begin{aligned} \begin{bmatrix} s' \\ d' \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ -\frac{\sqrt{3}}{4} + \frac{2-\sqrt{3}}{4} z^{-1} & 1 \end{bmatrix} \begin{bmatrix} s' \\ d \end{bmatrix} \\ &= \begin{bmatrix} s' \\ d + s' \cdot \frac{-\sqrt{3}}{4} + s' \cdot \frac{2-\sqrt{3}}{4} z^{-1} \end{bmatrix} \end{aligned} \tag{11}$$

which perform one multiplication and one addition in order to solve s' (as shown at the top resulting term in Eq. 10) and two multiplications and two additions to solve d' (as shown at the bottom resulting term in Eq. 11).

3.1 Architecture of the Processing Element

Taking into account that multipliers are expensive in term of area and the PE receives two samples (s and d) at once, we have decided to lower the input rate by half. From the performance point of view, the processing rate of the PE will be equal to the processor speed and no longer twice as fast. This also implies that the bottleneck issues on the input and output ports with the memory will not occur. From the hardware implementation point of view, the PE requires only one multiplier and one adder. This optimization, as detailed

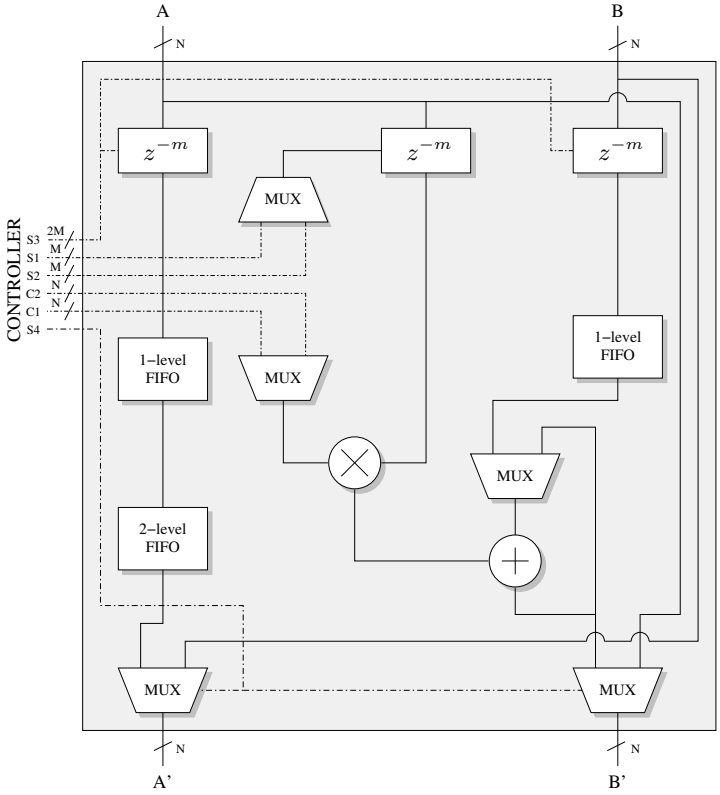

Fig. 3. Block diagram of the processing element.

later, is accomplished by multiplexing the operands of the multiplier inputs (the *multiplier* and the *multiplicand*) and by feeding the adder result back via the multiplexer.

Fig. 3 depicts the proposed PE. The PE has two selectors S1 and S2 to choose the prediction or the update samples that correspond to the factors p and q from the Laurent polynomial. Two constants which represent the filter coefficients are defined and configured by the controller. By delaying the actual samples, selector S3 controls the prediction or the update that requires future samples. Selector S4 is a bypass selector. Because lifting steps of the higher order wavelet filters may require distance prediction or update samples, the maximum depth of the unit delay z^{-m}, that determines the maximum delay level, can be freely chosen during the design.

Fig. 4 details the MAC (Multiply-and-Accumulate) unit which is implemented inside the PE. Both multiplier unit and adder unit require only one clock cycle to perform their function. C_1 and C_2 correspond to the Laurent polynomial constants, whereas M_1 and M_2 correspond to the outputs of the samples that are selected by S1 and S2. The multiplexer for M_1 and M_2 as

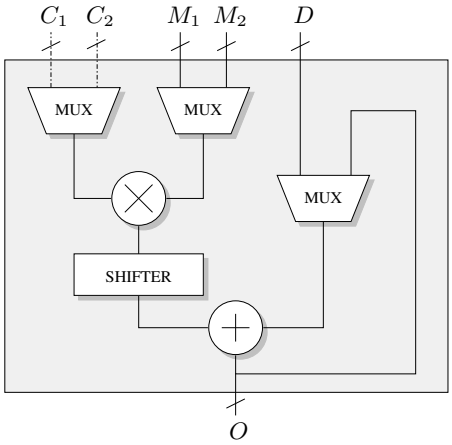

Fig. 4. Multiply-And-Accumulate unit.

a matter of fact does not exist and is drawn here only to illustrate the MAC process. A shifter is utilized as a replacement of the more expensive divider.

The PE is divided into 3 blocks. The first block organizes the input samples from both channels. The second block chooses the inputs of the multiplier and performs the multiplication. As mentioned earlier, the PE utilizes only one multiplier which is time-shared in order to perform two multiplications. The first clock cycle performs the first multiplication (i.e. $C_1 \times M_1$) and the second cycle performs the second multiplication (i.e. $C_2 \times M_2$). The third block performs the summation between the reference sample and the prediction/update values. Similar technique is applied here in order to utilize only one adder. As shown in Fig. 4, the first addition cycle performs $D + 2^{-R}(C_1 \times M_1)$ and the second addition cycle adds-up the first one with $2^{-R}(C_2 \times M_2)$. Whilst the input data are integer, the shifter performs the division on the multiplication result with 2^R where R can be freely chosen. Two 1-level FIFOs (First In First Out) are implemented to deal with the multiplier delay and a 2-level FIFO is implemented to compensate the delay which is introduced by the adder.

3.2 Normalization

As the multiplication with the normalization factor can take place at the end of the transform in case of forward DWT/DWP or at the beginning of the transform in case of inverse DWT/DWP, two special processing elements to handle this function are required. Although the normalization step is different compared to the prediction or the update step in a manner that both inputs s and d are multiplied with constants K and $1/K$ respectively, we know for sure that two multiplications take place. To perform this normalization step, we extend the functionality of the PEs that are located on the top and on the bottom of the proposed wavelet processor instead of implementing a dedicated

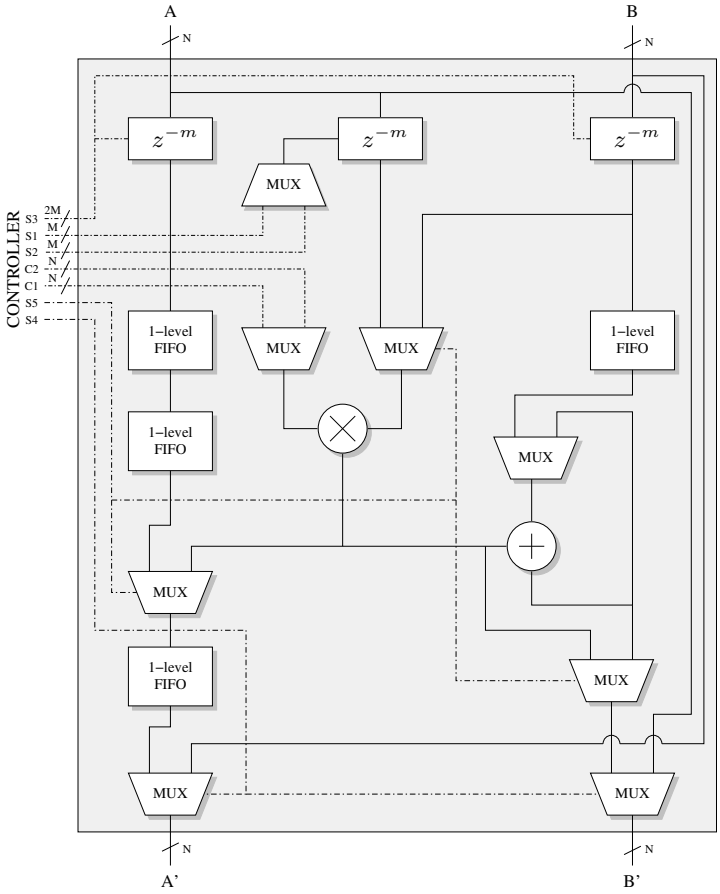

Fig. 5. Block diagram of the processing element which is located on the top and on the bottom of wavelet processor.

normalizer unit. Three additional multiplexers are needed to add the normalization factor unit into the PE. Fig. 5 shows the PE which is used on the top and on the bottom of the proposed architecture. By enabling S5 and setting S1 and S3 to zero, two inputs of the multiplexer before the multiplier correspond to the actual samples s and d (with the normalization factors $K = C_1$ and $1/K = C_2$). The first multiplication product passes through the multiplexer and the 1-level FIFO resulting $s' = Ks$ (the left side) and the second multiplication product passes through the multiplexer resulting $d' = d/K$ (the right side). Whereas the first normalization (i.e. $s' = Ks$) takes place first, instead of adding a 1-level FIFO on the right output port, the 2-level FIFO is split into two 1-level FIFOs to make both outputs synchronized and to minimize the latency.

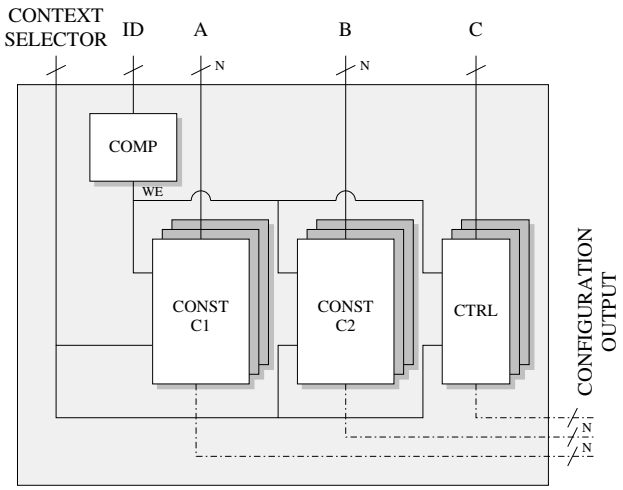

Fig. 6. Context Switch for the PE.

3.3 Context Switch

To cope with various lifting-based forward and inverse DWTs/DWPs, we have separated the configuration dependent parameters from the PE. Figs. 3 and 5 show how the inputs of the multiplexer selectors and the multiplier constants are separately drawn on the left side of the figures to emphasize the separation. In addition the PE is designed to be simple. Thus, no finite state machine is required to control the PE. To support different classes of wavelet filters that require different types of configurations, we have implemented a multi-context configuration on each PE as depicted in Fig. 6. Each PE is assigned with a row index as a unique ID for the configuration. Multiplier constants use the signal data paths to save the wiring cost whereas the multiplexers configuration requires additional controller path. Context switch is implemented as a memory module where the address is controlled by the context selector and the write enable signal is controlled by the output comparator.

The active configuration can easily be selected by using this context-based controller to cope with various wavelet filters. One benefit of having a multi-context configuration is that the proposed wavelet processor can be configured to perform the corresponding inverse DWTs/DWPs in a very simple manner. Additionally, the issues regarding the boundary condition can be relaxed by utilizing special wavelet filters on the signal boundaries which require less or no delayed/future samples (e.g. Haar wavelet) instead of exploiting the periodicity or the mirroring of the signal. Lastly, by using the context-based configuration, the DWTs/DWPs that exercise longer wavelet filters can simply be broken into smaller lifting steps. The configuration of each group of the lifting steps will be stored in the context memory and will be used to compute the transform.

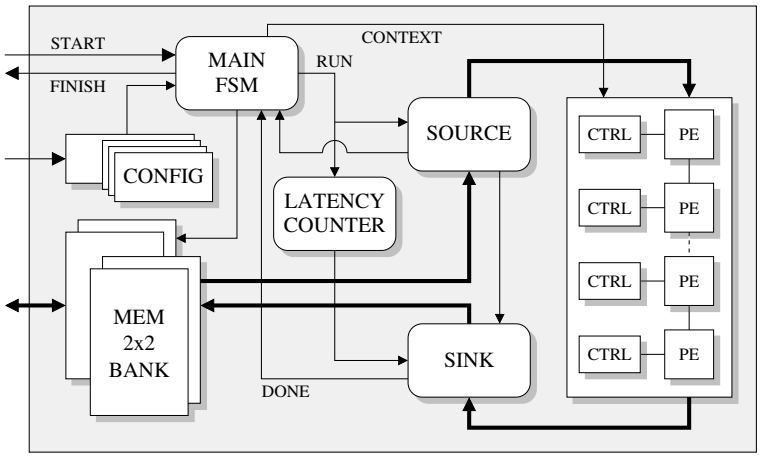

Fig. 7. The Proposed Wavelet Processor.

3.4 Memory Controller

Taking into account that the predictions and the updates occur alternately, the outputs of a PE will be cross-linked with the input of the next PE. Due to the nature of lifting steps, the prediction and the update are computed *in-place*. It means that it is not necessary to save the result or the temporary result into a different memory. One simple implementation of the proposed wavelet processor would consist of one PE. By configuring each context with the corresponding lifting step, the DWT/DWP and their inverses could be computed with this simple implementation. Although it is possible to use only one PE, a typical wavelet processor will have M chained PEs configuration to boost the performance and to minimize memory access.

Wavelet transform is a multi-resolutional signal processing tool. To achieve the required results, the signal needs to be transformed iteratively. In case of a DWT, only the low-pass part of the signal is taken into account as an input for the next transform. As a pair of low-pass and high-pass wavelet filter is used to compute the transform, the size of the signal decreases by two after each transformation level in this case. In contrary, a DWP uses both low-pass and high-pass parts of the signal in order to achieve equally spaced frequency bands after each transformation level. The total size of the signal on DWP remains the same and the amount of the processed data will slightly increase. It is due to the fact that low-pass and high-pass parts are treated independently during the computation and for each part of the signal, a signal extension, which will be detailed later on, is required to compute the transform on the boundary regions.

Fig. 7 depicts the block diagram of the processor along with the PEs and their configuration controller. The PEs that are located on the top and on

the bottom of the wavelet processor have an extra capability to perform the normalization.

Main FSM

The main finite state machine controls the wavelet processor. When the transform is initiated, the FSM reads the necessary configurations, such as the transformation level, forward/inverse mode, transform/packet mode, used contexts, etc. from the *config* block. This configuration, as detailed later, is divided into two categories. The first category is related to the functionalities of the processor and the second one is related to the lifting configuration.

The main FSM prepares the source and the sink addresses where the data will be read and stored, and also the length of the data needed to be processed. We exploit the periodicity extension to cope with the boundaries issue in order to compute the transform on those regions. This implies that source address does not always start on the top of the page. Address masking techniques are applied here to localize the page. The FSM takes care of the possibility of having a longer wavelet transform that has to be split into several lifting steps on the target PEs. The FSM allows multi-level forward/inverse DWT and DWP to take place by means of iteration process.

Config

The config block contains the configuration of the wavelet transform. Two different configuration categories are managed by this block. The functionality part manages:

- Selecting the type of the transform that will be performed: DWT or DWP.
- Selecting the transform mode: forward transform or inverse transform.
- The amount of memory that will be involved during the transform. Note that the processor can perform the transform on an arbitrary size of the sample. For an example, the value 0 indicates that the transform will be performed on the whole memory. The value 1 will make the transform processes the half of the memory and so on.
- Number of levels the transform will compute. This is effective to perform multi-level transform on a 1D signal. In contrary, for a 2D or higher dimension, the number of levels should always be set to 1.

The lifting part stores the configuration of the contexts used during the transform. It holds an important key to support wavelet transforms that use longer wavelet filters. If the number of lifting steps of the wavelet filters used for the transforms are larger than the available PEs, these lifting steps have to be split into several smaller steps that can be fit into the available PEs. The configuration of each lifting itself is stored on the context configuration of the PEs. This block stores only the corresponding context IDs that will be used. Thus, by selecting the right ID one after another, the wavelet transform

with longer lifting steps can be performed. Basically, it tells us which context should be used for the corresponding lifting step.

Beside storing the context IDs, it also holds the read and write offset addresses to start the transform and also the latency value for each lifting. It is important to note that in order to compute the wavelet transform, except for Haar wavelet filter, past and future samples are required. This becomes an issue when the transform on the signal boundary is performed. To cope with this boundary issue, the periodicity extension is used to locate these samples. These offsets hold the information of the corresponding starting sample for this periodicity extension.

Memory

The memory is organized as 2×2 banks. This configuration describes that the processor has two main banks (which are called bank 0 and bank 1) and each main bank consists of one primary bank and one shadow bank. With this technique, while the processor performs the transform on one bank (either bank 0 or bank 1), the next data can be placed on the other bank. Thus, it improves the overall performance by minimizing the delay caused by the data preparation.

The memory write and read accesses are exclusive, which means that writing to the memory will write to the primary bank and reading from the memory will read from its shadow. This state is switchable automatically, controlled by the *FSM*. When the transform takes place, the *FSM* grants the memory access of the selected bank to the *source* and *sink* blocks. Writing to or reading from this bank is forbidden and it will generate an error (as an indication of a busy signal). Nevertheless, the external interface can still read from and write to the memory of the other non-selected bank. Thus, the previous resulting transform, which is stored in this non-selected bank, can be read, and also the external interface can prepare the new data for the next transform.

Source and Sink

These blocks generate and automatically increment the read and write addresses. The *source* reads data from the memory and transfers it to the PEs. The *sink* reads data from the PEs and writes it to the memory. A special case is considered when performing transformations that are longer than the available PEs. During the in-between transformation, in case of forward transform, the *sink* will write the data (which corresponds to the intermediate results) to the memory in adjacent manner (resulting L-H-L-H-...). During the final transformation, the *sink* writes the LP and the HP signals into two different pages (resulting L-L-...-H-H-...). The similar handling is also performed by the *source* when performing the inverse transform.

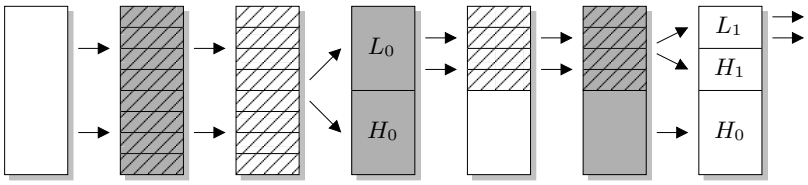

Fig. 8. Forward DWT Process.

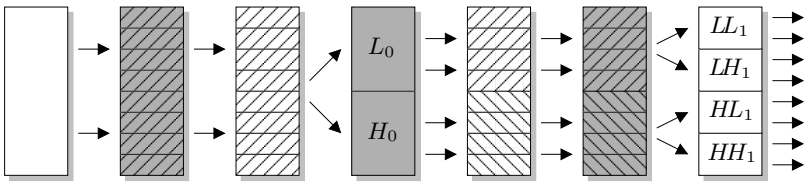

Fig. 9. Forward DWP Process.

To access the correct page, two address masks are used. The first mask is responsible for the data indexing, and the second mask is responsible for the page indexing.

Latency Counter

This block delays the *run* signal from the *main FSM* to initiate the sink process. The delay amount is different for every lifting steps and it is defined in the *config* block.

Details of the Memory Access

Fig. 8 illustrates the N-level and multiple lifting steps DWT. White and grey represent the primary and the shadow banks and diagonal pattern represents the in-between transformation. During the setup, the data is prepared and stored in one bank (this bank is write-only and its shadow is read-only). When the transformation is initiated, this state is reversed, and the source and the sink control the address lines. For each lifting steps, the source reads the written data, and the sink writes the in-between transformation result to the shadow bank. This state is reversed again every time one lifting step is finished, which makes the shadow bank as the primary bank and vice versa. During the last lifting step, the sink stores the LP and the HP results into two different pages. This whole process is performed N times with each iteration decreases the data by half. At each finishing level, a memory copy to transfer the previous HP result to the shadow bank is performed when necessary, e.g. when the lifting steps are odd.

For the DWPs, the HP signal is also transformed, as depicted in Fig. 9. Instead of executing/finalizing the transformation on each signal (LP, and

then HP) on each level, the in-between transformations are performed on both signals. With this technique, the banks are not switched during the in-between transformation for both LP and HP signals. Thus, the *FSM* can trigger the *source* to initiate the next data transfer for the next band/page (e.g. HP) without waiting the *sink* to finish from the previous transform. This solution decreases the data preparation time that is caused by exploiting the periodicity extension and the PEs latency. No copy transfer is performed on the DWPs/IDWPs.

4 Results and Performances

Our wavelet processor is written in VHDL and is based on the modular and parametric approach to make the design adaptable. In this paper, we provide the synthesis results of our wavelet processor that contains 8 PEs to process forward/inverse DWTs/DWPs with 8-level unit delays to support higher order wavelet filters and 16 available contexts to configure 16 different transforms. The design is synthesized using 0.18-μm technology. Because wavelet transforms deal with large numbers of samples, $2\times2\times512$ words memory is integrated into the processor for this implementation. Note that the wavelet processor is also designed to be flexible in respect with the number of the samples the processor can handle. In other word, the processor can be synthesized with an arbitrary size of the memory, as long as it follows an integer power of two rule. The size of the memory corresponds to the maximum number of samples the wavelet transforms can be performed by the processor.

The estimated area and frequency of various data width implementations are reported in Table 1. For the 16-bit configuration, the proposed wavelet processor consumes 2.5 mm^2 chip area and has a maximum operating speed of 319 MHz. As a comparison, architecture from Andra with 16-bit data width in [2] can only compute (5,3) and (9,7) filters and required 2.8 mm^2 with 200 MHz operating frequency. The details of the comparisons with the other architectures are summarized in Table 2. Note that our proposed architecture has flexible data width and memory size.

In order to realize the fixed-point multiplication between the samples and the coefficients, we utilized an integer multiplier and a shifter to reduce the hardware cost. As the compensation, this implementation leads to errors caused by the rounding of the wavelet coefficients and the cropping of the multiplication results. To measure the level of correctness of our design, we perform DWTs/DWPs and their corresponding inverse transforms on some predefined signals. Four different 8-bit full-swing signals, which are used as references, are forward and inverse transformed using Daub-4, Symlet-6, and Coiflet-2 wavelet filters with no integer coefficients. The random signal has a uniform distribution.

The lifting step coefficients of these wavelet filters are summarized in Table 3. These coefficients are shortened to save space. Because the coefficients

Table 1. Estimated area and frequency of proposed wavelet processor with 8 PEs and 2×2 memory banks.

Data Width	Est. Area (in mm^2)	% Area for Logic	Est. Frequency (in MHz)
16-bit	2.501	30.60 %	319.49
20-bit	3.120	31.38 %	298.51
24-bit	3.780	32.63 %	285.71
28-bit	4.376	32.57 %	262.47
32-bit	5.134	34.63 %	241.55

Table 2. Comparison with other Lifting-Based Architectures.

Arch.	Speed	Area	Filter	Transform	Data Width	Mem. Size
Andra [2]	200 MHz (0.18-μm)	2.8 mm^2	(5,3) (9,7)	DWT IDWT	16-bit	128
Dillen [13]	110 MHz (FPGA)	–	(5,3) (9,7)	DWT IDWT	16-bit	256
Seo [20]	150 MHz (0.35-μm)	5.6 mm^2	(5,3) (9,7)	DWT IDWT	12-bit	512
Wang [23]	100 MHz (0.18-μm)	1.1 mm^2	Daub-4	DWP	18-bit	8
Ours	242 MHz (0.18-μm)	5.1 mm^2	Arbitary	DWT,IDWT DWP,IDWP	32-bit* Configurable	512* Configurable

have to be represented as integers, depending on the data width, they will be magnified with some factor, and the result will be rounded and used as lifting coefficients. ModelSim is used to compare and verify the results. The SNR is computed using:

$$SNR_{(dB)} = 20 \times log_{10} \left(\frac{\sum |signal|}{\sum |signal - result|} \right) \tag{12}$$

where *signal* corresponds the input vector and *result* corresponds the output of the forward and inverse transforms.

Because wavelet transform is a multi-resolution signal processing tool, we perform four-level DWTs and DWPs to give a better overview of the performance of our wavelet processor. The SNR values of the different data width implementations for 4-level DWTs and DWPs are reported in Table 4 and Table 5 respectively. Depending on the data widths, SNR values vary between 29 dB and 140 dB in case of DWTs and between 27 dB and 138 dB in case of DWPs, which are sufficient for most applications. DWPs achieve slightly lower SNR values due to the fact that the high-pass signals after each transformation level get smaller and tend toward zero. Thus information losses are

Table 3. Lifting coefficients of Daub-6, Symlet-6, and Coiflet-2 wavelet filters.

Type	Daub-6	Symlet-6	Coiflet-2
Updater	2.425 z^0	-0.227 z^0	-2.530 z^0
Predictor	0.079 z^{-1} -0.352 z^0	-1.267 z^{-1} 0.216 z^0	-0.240 z^{-1} 0.342 z^0
Updater	-2.895 z^1 0.561 z^2	0.505 z^1 -4.255 z^2	3.163 z^1 15.268 z^2
Predictor	-0.020 z^{-2}	0.045 z^{-3} 0.233 z^{-2}	0.006 z^{-3} -0.065 z^{-2}
Updater		-18.389 z^3 6.624 z^4	-63.951 z^3 13.591 z^4
Predictor		0.144 z^{-5} -0.057 z^{-4}	0.001 z^{-5} 0.002 z^{-4}
Updater		-5.512 z^5	-3.793 z^5
Normalizer	0.432 2.315	-0.599 -1.671	0.108 9.288

Table 4. SNR values of different data width implementations (in dB) for 4-level forward and inverse DWT.

	Daub-6				
Source	16-bit	20-bit	24-bit	28-bit	32-bit
Sinusoid	42.90	67.04	89.38	115.00	138.52
Sawtooth	40.93	65.19	88.34	113.31	137.03
Step	44.98	67.07	87.95	114.19	138.88
Random	40.17	64.92	88.62	113.06	136.87
	Symlet-6				
Sinusoid	37.04	61.95	88.40	111.85	134.88
Sawtooth	35.75	60.22	85.84	108.89	133.17
Step	34.97	64.94	91.83	112.53	140.07
Random	36.52	61.18	85.93	109.37	133.51
	Coiflet-2				
Sinusoid	31.35	55.13	78.56	101.70	124.05
Sawtooth	29.80	52.85	76.83	100.13	123.19
Step	31.86	56.75	79.89	101.45	123.60
Random	29.01	52.83	77.53	101.93	125.27

affected at these bands. The 16-bit implementation achieves lower SNR values due to the fact that the lifting coefficients have a large dynamic range that is between 0.001 and 64. The same reason applies for Coiflet-2 wavelet filter. The improvement of the SNR values can be achieved by increasing the data width.

The proposed wavelet processor can accept input data stream and perform the computation in every two clock cycles made possible by the pipeline structure and the resource sharing. The total latency on each PE is 4 clock cycles. One clock cycle is consumed by the input registers, 1+1 by the multiplier (two

Table 5. SNR values of different data width implementations (in dB) for 4-level forward and inverse DWP.

	Daub-6				
Source	16-bit	20-bit	24-bit	28-bit	32-bit
Sinusoid	39.66	63.92	87.49	111.65	136.02
Sawtooth	37.45	62.05	85.26	109.80	134.00
Step	41.11	63.85	86.79	112.82	137.83
Random	37.19	61.75	84.11	109.34	133.41
	Symlet-6				
Sinusoid	35.41	60.03	85.22	108.95	131.86
Sawtooth	33.79	58.24	82.98	106.96	130.10
Step	34.25	62.31	87.14	108.17	134.37
Random	33.35	58.40	82.67	106.87	129.71
	Coiflet-2				
Sinusoid	29.26	53.07	76.74	100.13	123.01
Sawtooth	27.09	51.00	74.44	98.75	121.57
Step	29.49	53.75	76.65	98.84	122.46
Random	26.75	51.33	74.64	98.49	120.88

multiplications are performed), and 1+0 by the adder (two summations are performed where one cycle is "stolen" from the multiplier). Additional sample latency (2 clock cycles per future sample) will add-up to the total latency on the PEs which require this feature. The PE that is configured to perform the normalization step has latency of 3 clock cycles.

For the wavelet processor with M PEs, the total time needed to compute L-stage forward/inverse DWT is:

$$T_{DWT} = L(T_s + T_d) + 2S(1 - 0.5^L) + S(1 - 0.5^{L-1}) \tag{13}$$

where S is the signal length, T_s is the setup delay and $T_d = \sum_1^{m=M} T_{PE_m}$ is the circuit delay with T_{PE_m} as the PE latency delay of the m-th PE. The second term is the contribution of the actual transform whereas the last term is the result of the memory copy process.

In case of a L-stage forward/inverse DWP, the total time is formulated as:

$$T_{DWP} = L(T_s + T_d) + LS \tag{14}$$

The second term is the contribution of the low-pass and high-pass parts which have to be processed as well. No memory copy process takes place on performing forward/inverse DWP.

5 Conclusions

The facts are that wavelets have a very wide spectrum and there exists different classes of wavelet filters that can be used depending on the application. We have proposed a novel architecture that is able to compute various wavelet transforms and their inverses based on their lifting scheme representations. Because of diversities in application's need, we have designed the wavelet processor that can perform not only DWTs, but also DWPs.

The proposed wavelet processor is based on M chained PEs to compute the prediction/update of the lifting steps, and it can be configured easily to support higher order lifting polynomials, as the result of the factorization of the higher order wavelet filters. To cope with different wavelet filters, the developed wavelet processor includes a multi-context configuration so that users can easily switch between transforms (including their inverses). The wavelet processor is full-customized to manage different application demands which require different accuracy. Additionally, the architecture takes into account the energy conservation property of the wavelet transform by providing the normalization step that occurs at the end of the forward DWT/DWP or at the beginning of the inverse DWT/DWP. Due to its locality property, wavelet transform has a straightforward implementation in hardware. Considering also that wavelet transforms work with arbitrary number of samples, we deliver this freedom into our wavelet processor. Using 0.18-μm technology, the estimated area of the proposed wavelet processor with 16-bit configuration and $2\times2\times512$ words memory is 2.5 mm^2 and the estimated operating speed is 319 MHz.

References

1. J. Agbinya. Discrete wavelet transform techniques in speech processing. In *Proc. of the IEEE TENCON. Digital Signal Processing Applications, TENCON '96*, volume 2, pages 514–519, 1996.
2. K. Andra, C. Chakrabarti, and T. Acharya. A VLSI architecture for lifting-based forward and inverse wavelet transform. 50(4):966–977, 2002.
3. S. Barua, J. Carletta, K. Kotteri, and A. Bell. An Efficient Architecture for Lifting-based Two-Dimensional Discrete Wavelet Transforms. In *Proc. of the Great Lakes Symposium on VLSI, GLSVLSI '04*, 2004.
4. K. J. Blinowska and P. J. Durka. Introduction to wavelet analysis. *Br J Audiol*, 31(6):449–459, Dec 1997.
5. A. Bultheel. Wavelets with applications in signal and image processing, 2003.
6. R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo. Lossless image compression using integer to integer wavelet transforms. In *Proc. of the Intl. Conference on Image Processing, ICIP '97*, volume 1, pages 596–599. IEEE Press, 1997.
7. R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo. Wavelet transforms that map integers to integers. *Appl. Comput. Harmon. Anal.*, 5(3):332–369, 1998.

8. B. Carnero and A. Drygajlo. Perceptual speech coding and enhancement using frame-synchronized fast wavelet packet transform algorithms. 47:1622–1634, 1999.
9. C. Christopoulos, A. Skodras, and T. Ebrahimi. The JPEG2000 still image coding system: an overview. 46(4):1103–1127, 2000.
10. P. Dang and P. Chau. Reduce complexity hardware implementation of discrete wavelet transform for JPEG 2000 standard. In *Proc. of the IEEE Intl. Conference on Multimedia and Expo ICME '02*, volume 1, pages 321–324, 26–29 Aug. 2002.
11. I. Daubechies. The wavelet transform, time-frequency localization and signal analysis. *IEEE Trans. on Information Theory*, 36:961–1005, 1990.
12. I. Daubechies and W. Sweldens. Factoring Wavelet Transforms into Lifting Steps. *J. Fourier Anal. Appl.*, 4(3):245–267, 1998.
13. G. Dillen, B. Georis, J. Legat, and O. Cantineau. Combined line-based architecture for the 5-3 and 9-7 wavelet transform of JPEG2000. 13(9):944–950, Sept. 2003.
14. K. Ferens and W. Kinsner. Adaptive wavelet subband coding for music compression. In W. Kinsner, editor, *Proc. of the Data Compression Conference, DCC '95*, 1995.
15. S. Grgic, K. Kers, and M. Grgic. Image compression using wavelets. In K. Kers, editor, *Proc. of the IEEE Intl. Symposium on Industrial Electronics, ISIE '99*, volume 1, 1999.
16. Y. Kaisheng and C. Zhigang. A wavelet filter optimization algorithm for speech recognition. In *Intl. Conference on Communication Technology Proc., ICCT '98*, volume 2, page 5, 22-24 Oct. 1998.
17. S. Mallat, editor. *A Wavelet Tour of Signal Processing*. Academic Press, Incorporated, 1998.
18. M. Martina, G. Masera, G. Piccinini, and M. Zamboni. A VLSI architecture for IWT (Integer wavelet Transform). In G. Masera, editor, *Proc. of the 43rd IEEE Midwest Symposium on Circuits and Systems*, volume 3, 2000.
19. Y. Meyer. *Wavelets and Operators*. Press Syndicate of the University of Cambridge, 1992.
20. Y.-H. Seo and D.-W. Kim. A New VLSI Architecture of Lifting-Based DWT. *Lecture Notes in Computer Science*, 3985/2006:146–151, 2006.
21. W. Sweldens. The Lifting Scheme: A New Philosophy in Biorthogonal Wavelet Constructions. *Wavelet Applications in Signal and Image Processing*, 3:68–79, 1995.
22. B. Usevitch. A tutorial on modern lossy wavelet image compression: foundations of JPEG2000. 18(5):22–35, 2001.
23. C. Wang and W. Gan. Efficient VLSI Architecture for Lifting-Based Discrete Wavelet Packet Transform. 54(5):422–426, 2007.