Circuits Syst Signal Process (2012) 31:987–1015
DOI 10.1007/s00034-011-9355-0

FPGA-Implementation of Discrete Wavelet Transform with Application to Signal Denoising

Mohammed Bahoura · Hassan Ezzaidi

Received: 21 December 2010 / Revised: 15 August 2011 / Published online: 21 September 2011

Abstract This paper presents new architectures for real-time implementation of the forward/inverse discrete wavelet transforms and their application to signal denoising. The proposed real-time wavelet transform algorithms present the advantage to ensure perfect reconstruction by equalizing the filter path delays. The real-time signal denoising algorithm is based on the equalized filter paths wavelet shrinkage, where the noise level is estimated using only few samples. Different architectures of these algorithms are implemented on FPGA using Xilinx System Generator for DSP and XUP Virtex-II Pro development board. These architectures are evaluated and compared in terms of reconstruction error, denoising performance and resource utilization.

Keywords Wavelet transform · Signal denoising · Soft-thresholding · Filter group delay · Pipelining · FPGA

M. Bahoura (✉)
Department of Engineering, Université du Québec à Rimouski, 300, allée des Ursulines, Rimouski, QC, G5L 3A1, Canada
e-mail: Mohammed_Bahoura@uqar.qc.ca

H. Ezzaidi
Department of Applied Sciences, Université du Québec à Chicoutimi, 550, boul. de l'Université, Chicoutimi, QC, G7H 2B1, Canada
e-mail: hezzaidi@uqac.ca

1 Introduction

In the last decades, the wavelet transform has been successfully used in wide range of applications across several disciplines, including signal and image denoising and compression, feature extraction, signal detection, pattern recognition, etc. The extensive use of the wavelet transform can be explained by its capability to provide simultaneous time-scale analysis and its indefinite number of basis functions.

Due to recent advances in technology and decreasing costs, implementation of the discrete wavelet transform (DWT) on field-programmable gate array (FPGA) has been widely developed. To optimize time and resources consuming, many FPGA architectures of the DWT have been proposed, which are mainly based on convolution [8, 11, 17] and lifting [1, 16, 23] schemas. To our knowledge, these architectures do not take into account the group delays of the finite impulse response (FIR) filters that are used to compute the DWT. These delays do not have a consequence in image processing or when the signal is processed frame by frame. However, when the signal is processed sample by sample, the group delays of different filter paths affect considerably the synchronism between the wavelet coefficients in various scale levels, which is aggravated by scale level increasing. Consequently, the reconstruction of the signal from the wavelet coefficients will be altered because the delays will be accumulated.

Wavelet-based denoising techniques were successfully applied to speech signal [5, 6], electrocardiogram (ECG) [18, 21], encephalogram [19], image [7], etc. Real-time implementation of these algorithms using FPGA constitutes a great challenge, where these techniques must be reorganized to meet the time and material constraints of the existing technology. However, a real-time wavelet-based signal denoising system is generally based on a real-time analysis/synthesis architecture that guaranties perfect reconstruction of the signal.

In this paper, we propose a real-time implementation of the forward/inverse wavelet transforms and their application to signal denoising. The proposed architectures have been implemented on FPGA using Xilinx System Generator and XUP Virtex-II Pro development board. The rest of the paper is organized as follows. Section 2 presents the proposed architectures for the real-time wavelet analysis/synthesis transforms and their application to signal denoising. Section 3 describes in detail the hardware implementation and the resource requirement of these architectures. Experimental results are presented and discussed in Sect. 4. Finally, conclusion is given in Sect. 5.

2 Method

This section describes the proposed real-time architectures. The first subsection presents the wavelet transform theory and its implementation using the classical frame-based and the proposed sample-based real-time algorithms. The second subsection presents the usual frame-based and the proposed sample-based signal denoising algorithms. For both wavelet analysis/synthesis and signal denoising algorithms, different architectures are defined depending on the analysis/synthesis schema, the canonical form of the wavelet-based filters and the conventional or pipelined configuration.

2.1 Wavelet Transform

2.1.1 Continuous Wavelet Transform

The continuous wavelet transform of a signal $x(t)$ is defined by

$$w_x^{\psi}(s,b) = \int_{-\infty}^{\infty} x(t)\psi_{s,b}^*(t)\,dt, \tag{1}$$

where $(^*)$ denotes the complex conjugate, and $\psi_{s,b}(t)$ are the basis functions obtained by dilation or contraction (scaling), and translation of the mother wavelet $\psi(t)$:

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

where $s \in \mathbb{R}^+$ represents the scale parameter, and $b \in \mathbb{R}$ represents the time translation. A large value of the parameter s stretches the basis wavelet function and allows the analysis of low-frequency components of the signal. On the other hand, a small value of this parameter contracts this function and allows the analysis of the high-frequency components [21].

The inverse continuous wavelet transform is obtained using

$$x(t) = \frac{1}{C_{\psi}} \int_0^{\infty}\int_{-\infty}^{\infty} w_x^{\psi}(s,b)\psi_{s,b}\frac{ds\,db}{s^2}, \tag{3}$$

where C_{ψ} is a normalizing parameter that depends on $\psi(t)$.

2.1.2 Discrete Wavelet Transform

The discrete wavelet transform (DWT) is obtained by discretizing the parameters s and b. In general, the scale parameter is discretized by an exponential sampling with fixed step, $s = s_0^j$, and the translation parameter by integer multiple of a scale dependent step, $b = kb_0s$, where $s_0 > 1$, $b_0 > 0$, and $k \in \mathbb{Z}$. Particularly, one can chose samples from dyadic grid: $s_0 = 2$ and $b_0 = 1$. The DWT is then defined as

$$w_j[k] = 2^{-\frac{j}{2}} \int_{-\infty}^{\infty} x(t)\psi^*\left(2^{-j}t - k\right)dt. \tag{4}$$

Here 2^j represents the resolution or scale, j is the corresponding resolution level, and k is the shifting parameter.

The discrete wavelet transform (DWT) is usually computed using the pyramid algorithm proposed by Mallat [13]. As presented in Fig. 1, this algorithm is based on a pair of high-pass (G) and low-pass (H) filters, named also quadrature mirror filters (QMF), that are related through

$$g[n] = (-1)^{n+1}h[N-n-1], \tag{5}$$

where $g[n]$ and $h[n]$ are the impulse responses of G and H, respectively, and N is the number of their coefficients.

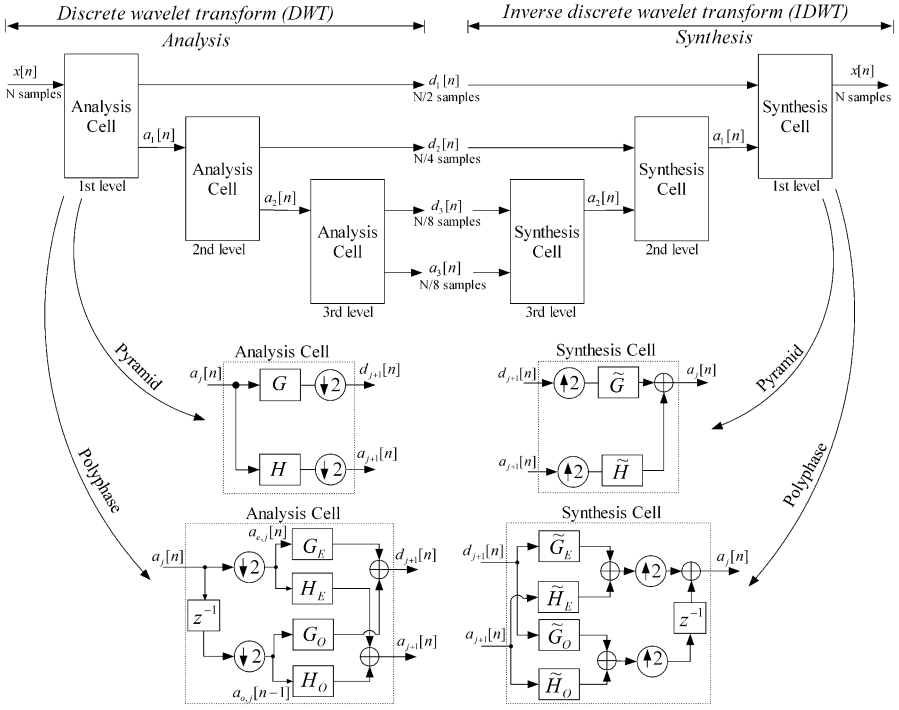


Fig. 1 Diagram of a frame-based forward and inverse wavelet transforms. Details of analysis/synthesis cells are also given for pyramid and polyphase architectures

These filters are constructed from the wavelet kernel $\psi(t)$ and its companion scaling function $\phi(t)$. They are related by the following relations:

$$\phi(t) = \sqrt{2}\sum_k h[k]\phi(2t-k), \tag{6}$$

$$\psi(t) = \sqrt{2}\sum_k g[k]\phi(2t-k). \tag{7}$$

The outputs of the high-pass filters are named details $d_j[n]$, and those of the low-pass filters are named approximations $a_j[n]$. They are defined by

$$d_{j+1}[n] = \sum_k a_j[k]g[2n-k], \tag{8}$$

$$a_{j+1}[n] = \sum_k a_j[k]h[2n-k]. \tag{9}$$

It can be noted that (8) and (9) represent the analysis cell of the pyramid architecture that is described within a dashed box in Fig. 1.

The inverse discrete wavelet transform (IDWT) uses a pair of low-pass, $\widetilde{h}[n]$, and high-pass, $\widetilde{g}[n]$, reconstruction filters. The analysis and synthesis filters are related to

each other through

$$\widetilde{g}[n] = g[N-n-1], \tag{10}$$

$$\widetilde{h}[n] = h[N-n-1]. \tag{11}$$

The reconstructed approximation is given by

$$a_j[n] = \sum_k \big(d_{j+1}[k]\widetilde{g}[n-2k] + a_{j+1}[k]\widetilde{h}[n-2k]\big). \tag{12}$$

Equation (12) represents the synthesis cell of the pyramid architecture that is also described within a dashed box in Fig. 1. However, only orthogonal wavelets allow for perfect reconstruction of a signal by IDWT.

It can be seen that for the pyramid architecture, the half of mathematical computation are wasted [12]. In the analysis cell, down-sampling discards half of samples computed by filters; in the synthesis cell, up-sampling before filtering means that half of the filter multiplications is done with the inserted zeros [12]. To overcome this drawback, the polyphase structure has been proposed. Wasted operations are avoided by placing down-sampling before filtering in the analysis cell and up-sampling after filtering in the synthesis cell (Fig. 1). For a given level analysis cell of the pyramid architecture, the detail output $d_{j+1}[n]$, defined by (8), can be expressed as [22]

$$\begin{aligned} d_{j+1}[n] &= \sum_{k=0}^{\frac{N}{2}-1} g[2k]a_j[2n-2k] + \sum_{k=0}^{\frac{N}{2}-1} g[2k+1]a_j[2n-2k-1] \\ &= \sum_{k=0}^{\frac{N}{2}-1} g_e[k]a_{e,j}[n-k] + \sum_{k=0}^{\frac{N}{2}-1} g_o[k]a_{o,j}[n-k-1], \end{aligned} \tag{13}$$

where $g_e[k]$ and $g_o[k]$ are the even and odd coefficients of the filter G, respectively. The input signal $x[n]$ is split into even samples $x_e[n]$ and odd samples $x_o[n]$. On the other hand, the approximation output $a_{j+1}[n]$, defined by (9), can be expressed as

$$\begin{aligned} a_{j+1}[n] &= \sum_{k=0}^{\frac{N}{2}-1} h[2k]a_j[2n-2k] + \sum_{k=0}^{\frac{N}{2}-1} h[2k+1]a_j[2n-2k-1] \\ &= \sum_{k=0}^{\frac{N}{2}-1} h_e[k]a_{e,j}[n-k] + \sum_{k=0}^{\frac{N}{2}-1} h_o[k]a_{o,j}[n-k-1], \end{aligned} \tag{14}$$

where $h_e[k]$ and $h_o[k]$ are the even and odd coefficients of the filter H, respectively. Hence, (13) and (14) define the analysis cell of the polyphase architecture, which is also described within a dashed box in Fig. 1. Similar development can be followed to define the synthesis cell.

Despite the fact that polyphase structure is much more efficient than pyramid structure, as it reduces the number of multiplication operations by half, these architectures require the same number of multipliers.

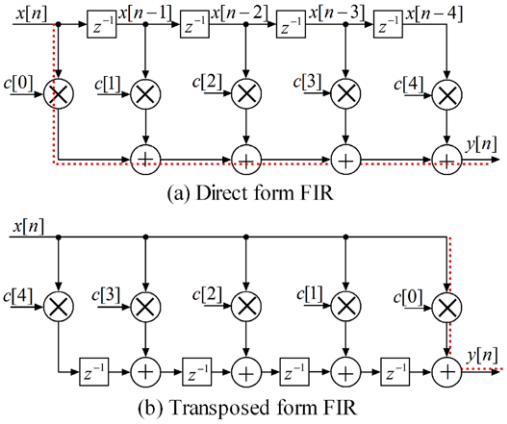


Fig. 2 Canonical implementation of a five-tap FIR filter. The critical path is illustrated by the *dashed line*

2.1.3 Canonical Forms of the Wavelet-Based FIR Filters

As described in the previous subsection, the forward and inverse discrete wavelet transforms are implemented using finite impulse response (FIR) filters both for pyramid and polyphase architectures. The output $y[n]$ of a FIR filter is defined as the convolution of N filter coefficients $c[k]$ with a sequence of input data samples $x[n]$:

$$y[n] = \sum_{k=0}^{N-1} c[k]x[n-k], \tag{15}$$

where $N-1$ is the filter order. N is the number of coefficients for the FIR filter, which is also defined as the number of taps.

The FIR filters can be implemented using two well-known canonical forms, called the direct and transposed forms [2, 4]. Figure 2 presents a five-tap FIR filters that are functionally equivalent. In real-world applications, both forms suffer from important drawbacks. The direct form is limited by the critical path corresponding to the longest computation time among all paths that contain zero delays. The critical path is an increasing function of the number of taps and at the same time needs to be less than a clock period. It is defined as $\tau_{cp} = \tau_m + (N-1)\tau_a$, where τ_m and τ_a are the times needed for one multiplication and one addition, respectively, and N is the number of taps. The transposed form overcomes this limitation by retiming delays into the adder chain. In this case, the critical path is reduced to a single multiply-accumulate operation, $\tau_{cp} = \tau_m + \tau_a$. However, this form suffers from significant fan-in to apply simultaneously the input data signal to all taps of the filter [2, 4].

2.1.4 Conventional Real-Time Wavelet Transform

Figure 1 presents the frame-based forward and inverse wavelet transforms architecture. When the input signal is processed frame by frame, the delays caused by the wavelet analysis/synthesis filters are not needed to ensure perfect reconstruction of the signal. However, when signal is processed sample by sample, these delays must

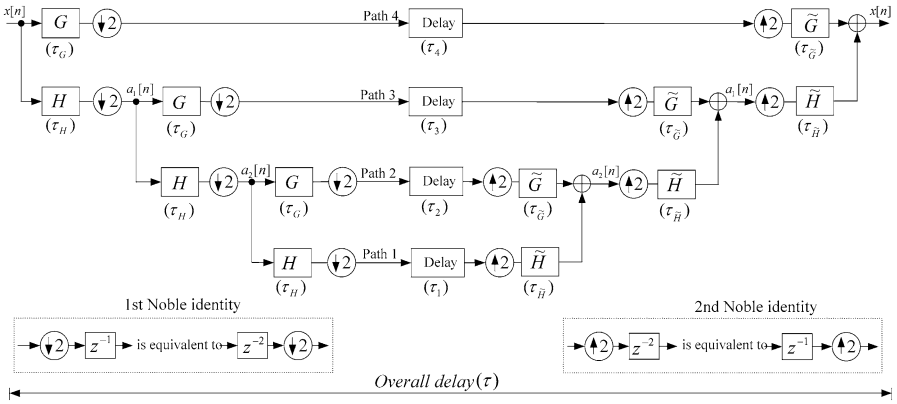


Fig. 3 Diagram of the proposed conventional real-time algorithm of a 3-level pyramid forward and inverse wavelet transforms [3]. The Noble identities are defined in the bottom [15]

be taken into account to guarantee perfect reconstruction. This can be achieved by equalizing delays over all filter paths [3, 15]. To our knowledge, these delays are neglected in all other published architectures for FPGA-implementation of DWT.

The proposed real-time architecture for pyramid forward and inverse wavelet transforms that ensure perfect reconstruction is presented Fig. 3. Equalization of the filter path delays is obtained by inserting appropriate additional delays in various paths.

For a given filter H, the group delay is defined as the negative derivative of the phase response versus frequency:

$$\tau_H(\omega) = -\frac{d\theta(\omega)}{d\omega}, \tag{16}$$

where $\theta(\omega)$ is the phase of the filter $H(e^{j\omega}) = |H(e^{j\omega})|e^{j\theta(\omega)}$.

The group delay is a measure of the relative delay at different frequencies from the input to the output in the filter. This parameter is constant for a linear phase filter.

Let τ_H, τ_G, $\tau_{\widetilde{H}}$ and $\tau_{\widetilde{G}}$ be the group delays of the filters H, G, $\widetilde{H}$ and $\widetilde{G}$, respectively. The delay of each path is obtained by summing delays of the cascaded filters using the Noble identities [15]:

- Path 1: $\tau = \tau_H + 2\tau_H + 4\tau_H + 8\tau_1 + 4\tau_{\widetilde{H}} + 2\tau_{\widetilde{H}} + \tau_{\widetilde{H}} = 7(\tau_H + \tau_{\widetilde{H}}) + 8\tau_1$
- Path 2: $\tau = \tau_H + 2\tau_H + 4\tau_G + 8\tau_2 + 4\tau_{\widetilde{G}} + 2\tau_{\widetilde{H}} + \tau_{\widetilde{H}}$
  $= 3(\tau_H + \tau_{\widetilde{H}}) + 4(\tau_G + \tau_{\widetilde{G}}) + 8\tau_2$
- Path 3: $\tau = \tau_H + 2\tau_G + 4\tau_3 + 2\tau_{\widetilde{G}} + \tau_{\widetilde{H}} = (\tau_H + \tau_{\widetilde{H}}) + 2(\tau_G + \tau_{\widetilde{G}}) + 4\tau_3$
- Path 4: $\tau = \tau_G + 2\tau_4 + \tau_{\widetilde{G}} = (\tau_G + \tau_{\widetilde{G}}) + 2\tau_4$.

They are expressed in samples. If $\tau_H + \tau_{\widetilde{H}} = \tau_G + \tau_{\widetilde{G}}$, these delays are redefined as:

- Path 1: $\tau = 7(\tau_H + \tau_{\widetilde{H}}) + 8\tau_1$
- Path 2: $\tau = 7(\tau_H + \tau_{\widetilde{H}}) + 8\tau_2$

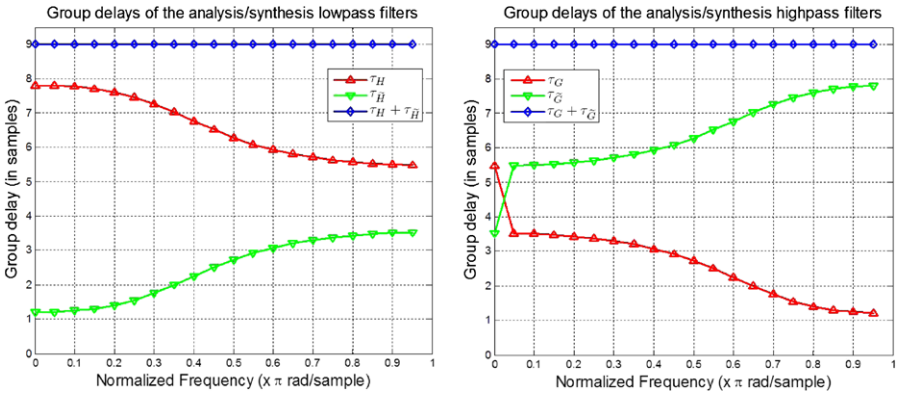


Fig. 4 Group delays of the analysis/synthesis filters (based on Daubechies 5 wavelet). The *left subfigure* presents the group delays of individual and cascaded analysis and synthesis low-pass filters, while the *right* one presents those of the high-pass filters

– Path 3: $\tau = 3(\tau_H + \tau_{\widetilde{H}}) + 4\tau_3$
– Path 4: $\tau = (\tau_H + \tau_{\widetilde{H}}) + 2\tau_4$.

Equalizing Path 1 and Path 2 to their minimum common value, $\tau = 7(\tau_H + \tau_{\widetilde{H}})$, leads to $\tau_1 = \tau_2 = 0$. By equalizing Path 3 and Path 4 to Path 1, one can deduce that $\tau_3 = (\tau_H + \tau_{\widetilde{H}})$ and $\tau_4 = 3(\tau_H + \tau_{\widetilde{H}})$.

Figure 4 presents the group delays τ_H, $\tau_{\widetilde{H}}$ and $\tau_H + \tau_{\widetilde{H}}$ on the left subfigure and τ_G, $\tau_{\widetilde{G}}$ and $\tau_G + \tau_{\widetilde{G}}$ on the right one. It can be shown that $\tau_H + \tau_{\widetilde{H}} = \tau_G + \tau_{\widetilde{G}} = 9$. In this case, the overall delay is given by $\tau = 7(\tau_H + \tau_{\widetilde{H}}) = 63$. The inserted delays τ_1, τ_2, τ_3 and τ_4 are equal to 0, 0, 9 and 27, respectively. The same delays are needed in the polyphase equivalent architecture. Coefficients and group delays of the wavelet filters are computed using *wfilters* and *grpdelay* functions of Matlab.

2.1.5 Pipelined Real-Time Wavelet Transform

Pipelining reduces the critical path delay by inserting delays between the computational elements (multipliers and adders) of the circuit. However, this process presents two main drawbacks: increasing the number of latches and the output latency in the system.

Figure 5 presents the canonical realizations of a five-tap pipelined FIR filter. The critical path is reduced to a single multiplication operation $\tau_{cp} = \tau_m$. It can be seen that the pipelining produces latency at the filter output, which is negligible for transposed form (latency $= 2$) but can be considerable for direct form because it is an increasing function of the number of taps (latency $= N$).

The proposed pipelined architecture for a 3-level forward and inverse wavelet transforms is presented in Fig. 6. In addition to the pipelining of the wavelet-based FIR filters, additional delays are inserted after the remaining adders in the analysis and synthesis cells. In this fully pipelined architecture, the critical path is reduced to a single multiplication operation.

The wavelet analysis/synthesis algorithm can be implemented in different ways, depending on the used schema (pyramid or polyphase), the canonical form of the FIR

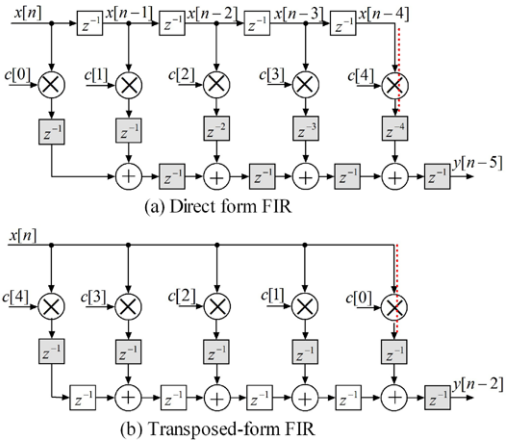


Fig. 5 Pipelined implementation of canonical forms presented in Fig. 2. The inserted delays are presented by the *shaded boxes*

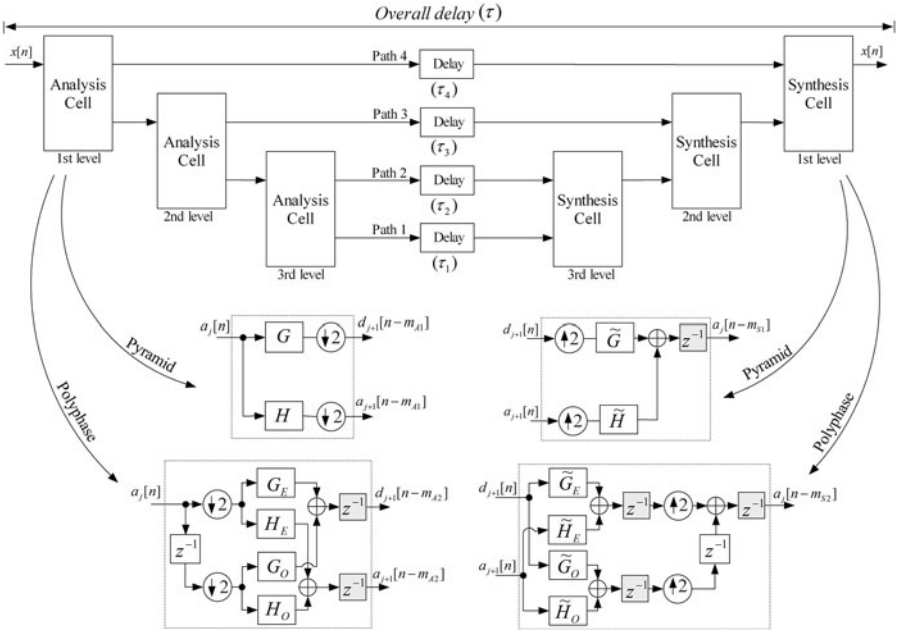


Fig. 6 Diagram of the proposed pipelined real-time architecture of a 3-level forward and inverse wavelet transforms. The pipelined wavelet-based FIR filters of the analysis/synthesis cells are detailed in Fig. 5

filters and the pipelining choice. Table 1 presents the abbreviations used to define the eight possible combinations implemented in this work. The filter path delays of these architectures are equalized by inserting appropriate delays to ensure perfect reconstruction. For the pipelined architecture of the wavelet analysis/synthesis algorithm that uses pyramid schema and direct-form FIR filters (AS-P-PY-DF), the path delays are obtained as follows:

Table 1 Abbreviations used to define the wavelet analysis/synthesis architectures

Abbreviation	Definition
AS-C-PY-DF	Wavelet analysis/synthesis using conventional pyramid algorithm and direct-form FIR
AS-C-PY-TF	Wavelet analysis/synthesis using conventional pyramid algorithm and transposed-form FIR
AS-C-PO-DF	Wavelet analysis/synthesis using conventional polyphase algorithm and direct-form FIR
AS-C-PO-TF	Wavelet analysis/synthesis using conventional polyphase algorithm and transposed-form FIR
AS-P-PY-DF	Wavelet analysis/synthesis using pipelined pyramid algorithm and direct-form FIR
AS-P-PY-TF	Wavelet analysis/synthesis using pipelined pyramid algorithm and transposed-form FIR
AS-P-PO-DF	Wavelet analysis/synthesis using pipelined polyphase algorithm and direct-form FIR
AS-P-PO-TF	Wavelet analysis/synthesis using pipelined polyphase algorithm and transposed-form FIR

– Path 1: $\tau = (\tau_H + N) + 2(\tau_H + N) + 4(\tau_H + N) + 8\tau_1 + 4(\tau_{\widetilde{H}} + N + 1) + 2(\tau_{\widetilde{H}} + N + 1) + (\tau_{\widetilde{H}} + N + 1) = 7(\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 8\tau_1$
– Path 2: $\tau = (\tau_H + N) + 2(\tau_H + N) + 4(\tau_G + N) + 8\tau_1 + 4(\tau_{\widetilde{G}} + N + 1) + 2(\tau_{\widetilde{H}} + N + 1) + (\tau_{\widetilde{H}} + N + 1) = 3(\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 4(\tau_G + \tau_{\widetilde{G}} + 2N + 1) + 8\tau_2$
– Path 3: $\tau = (\tau_H + N) + 2(\tau_G + N) + 4\tau_3 + 2(\tau_{\widetilde{G}} + N + 1) + (\tau_{\widetilde{H}} + N + 1) = (\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 2(\tau_G + \tau_{\widetilde{G}} + 2N + 1) + 4\tau_3$
– Path 4: $\tau = (\tau_G + N) + 2\tau_4 + (\tau_{\widetilde{G}} + N + 1) = (\tau_G + \tau_{\widetilde{G}} + 2N + 1) + 2\tau_4.$

If $\tau_H + \tau_{\widetilde{H}} = \tau_G + \tau_{\widetilde{G}}$, these delays are redefined as:

– Path 1: $\tau = 7(\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 8\tau_1$
– Path 2: $\tau = 7(\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 8\tau_2$
– Path 3: $\tau = 3(\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 4\tau_3$
– Path 4: $\tau = (\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 2\tau_4.$

Equalizing Path 1 and Path 2 to their minimum common value, $\tau = 7(\tau_H + \tau_{\widetilde{H}} + 2N + 1)$, leads to $\tau_1 = \tau_2 = 0$. By equalizing Path 3 and Path 4 to Path 1, one can deduce that $\tau_3 = (\tau_H + \tau_{\widetilde{H}} + 2N + 1)$ and $\tau_4 = 3(\tau_H + \tau_{\widetilde{H}} + 2N + 1)$.

For Daubechies 5 wavelet (db5), the analysis/synthesis filters have 10 coefficients each ($N = 10$). Delays on Path 1 and Path 2 are identical and equal to the overall delay $\tau = 7(9 + 2 \times 10 + 1) = 210$. The inserted delays are $\tau_1 = \tau_2 = 0$, $\tau_3 = 30$ and $\tau_4 = 90$.

Table 2 presents the inserted delays and the output latency of the wavelet analysis/synthesis architectures. As can be expected, the inserted delays τ_3 and τ_4 of the pipelined architectures are increased compared to those of the conventional ones, due to the delays added during the pipelining process. These delays are more important with the direct form than with the transposed form. Also, it can be seen that the conventional architectures present the same output latency of 63 samples. The output latencies of the pipelined architectures are more important than those of the

Table 2 Inserted delay values, expressed in samples, for eight wavelet analysis/synthesis architectures. The output latency is evaluated for 3-level schema using Daubechies 5 wavelet, i.e., $\tau_H + \tau_{\widetilde{H}} = 9$ and $N = 10$

Architecture	τ_1	τ_2	τ_3	τ_4	τ	Latency
AS-C-PY-DF	0	0	$\tau_H + \tau_{\widetilde{H}}$	$3(\tau_H + \tau_{\widetilde{H}})$	$7(\tau_H + \tau_{\widetilde{H}})$	63
AS-C-PY-TF	0	0	$\tau_H + \tau_{\widetilde{H}}$	$3(\tau_H + \tau_{\widetilde{H}})$	$7(\tau_H + \tau_{\widetilde{H}})$	63
AS-C-PO-DF	0	0	$\tau_H + \tau_{\widetilde{H}}$	$3(\tau_H + \tau_{\widetilde{H}})$	$7(\tau_H + \tau_{\widetilde{H}})$	63
AS-C-PO-TF	0	0	$\tau_H + \tau_{\widetilde{H}}$	$3(\tau_H + \tau_{\widetilde{H}})$	$7(\tau_H + \tau_{\widetilde{H}})$	63
AS-P-PY-DF	0	0	$\tau_H + \tau_{\widetilde{H}} + 2N + 1$	$3(\tau_H + \tau_{\widetilde{H}} + 2N + 1)$	$7(\tau_H + \tau_{\widetilde{H}} + 2N + 1)$	210
AS-P-PY-TF	0	0	$\tau_H + \tau_{\widetilde{H}} + 5$	$3(\tau_H + \tau_{\widetilde{H}} + 5)$	$7(\tau_H + \tau_{\widetilde{H}} + 5)$	98
AS-P-PO-DF	0	0	$\tau_H + \tau_{\widetilde{H}} + 2(N + 2) + 1$	$3(\tau_H + \tau_{\widetilde{H}} + 2(N + 2) + 1)$	$7(\tau_H + \tau_{\widetilde{H}} + 2(N + 2) + 1)$	238
AS-P-PO-TF	0	0	$\tau_H + \tau_{\widetilde{H}} + 13$	$3(\tau_H + \tau_{\widetilde{H}} + 13)$	$7(\tau_H + \tau_{\widetilde{H}} + 13)$	154

conventional one, where the largest value (238 samples) is obtained for AS-P-PO-DF architecture.

2.2 Wavelet Shrinkage

The wavelet shrinkage is a signal denoising technique that consists to apply a thresholding function to the wavelet coefficients. The main idea of this algorithm is that the wavelet coefficients with magnitudes smaller than the preset threshold are caused by noise and replaced by zero, and the others with amplitudes larger than the preset threshold are caused by the original signal mainly and kept (hard-thresholding case) or shrunk (soft-thresholding case) [20]. Then the denoised signal could be obtained from the resulting wavelet coefficients.

2.2.1 Principle

Donoho [9] proposed an original algorithm to recover a signal $x[n]$ by thresholding the wavelet coefficients of the observed signal $y[n]$,

$$y[n] = x[n] + b[n], \quad n = 0, \ldots, N-1, \tag{17}$$

where N is the frame length of the observed signal, and $b[n]$ is an additional Gaussian white noise. If the dyadic algorithm proposed by Mallat [13] is used, N must be a power of 2.

The wavelet-based denoising algorithm can be summarized in three steps:

- Wavelet transform of the noisy signal,
- Thresholding the resulting wavelet coefficients,
- Transformation back to obtain the cleaned signal.

Donoho and Johnstone [10] define the *soft*-thresholding function by

$$T_\lambda^S(w) = \begin{cases} 0 & \text{if } |w| \le \lambda, \\ w - \lambda & \text{if } w > \lambda, \\ w + \lambda & \text{if } w < -\lambda, \end{cases} \tag{18}$$

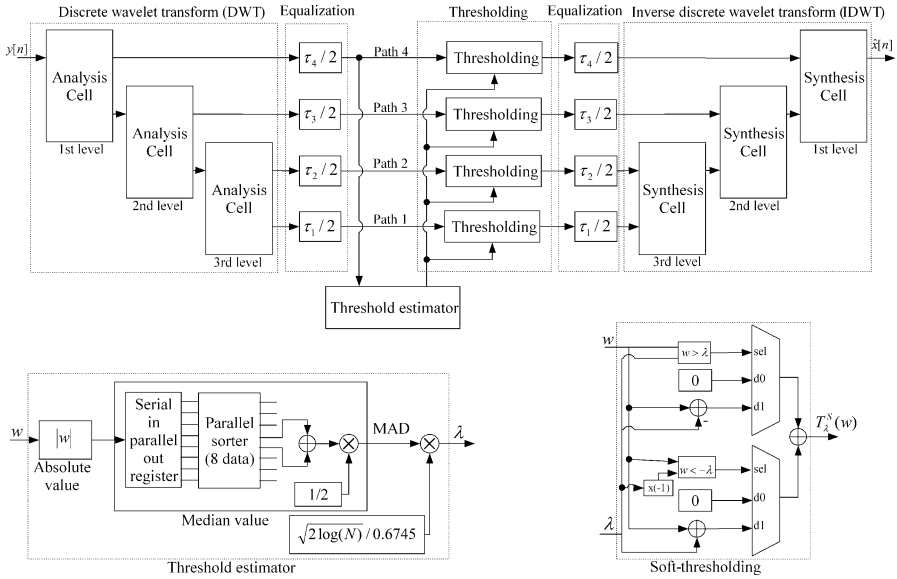


Fig. 7 *Diagram* of the proposed conventional real-time architecture of a wavelet-based denoising algorithm. The conventional threshold estimator and the thresholding blocks are detailed in the *bottom subfigures*

where $\lambda \geq 0$ is the threshold, and w represents a wavelet coefficient to be thresholded.

They proposed a universal threshold λ for the WT:

$$\lambda = \sigma\sqrt{2\log(N)} \tag{19}$$

with $\sigma = \mathrm{MAD}/0.6745$, where N is the length of the analyzed signal $y[n]$, and σ is the noise level. MAD is the median absolute deviation estimated on the first scale:

$$\mathrm{MAD} = \mathrm{median}\big(\big|w - \mathrm{median}(w)\big|\big). \tag{20}$$

The MAD is generally approximated by $\mathrm{median}(|w|)$.

2.2.2 *Conventional Real-Time Wavelet-Based Denoising*

The proposed architecture is based on the Donoho and Johnstone algorithm described in the previous subsection. However, this algorithm is reorganized to meet the real-time condition in which the signal is processed sample by sample unlike frame by frame (Fig. 7). This architecture is based on the proposed conventional real-time wavelet analysis/synthesis diagram of (Fig. 3) and additional blocks for the threshold estimation and the wavelet coefficients thresholding. Figure 7 also presents an implementation of the soft thresholding function (18). The universal threshold is applied to the detail coefficients of each level.

The noise level estimation constitute the main challenge in computing the universal threshold (19). This can be done by computing the MAD (20) of the wavelet coefficients at the first level. We recall that the median value of a set of M data is

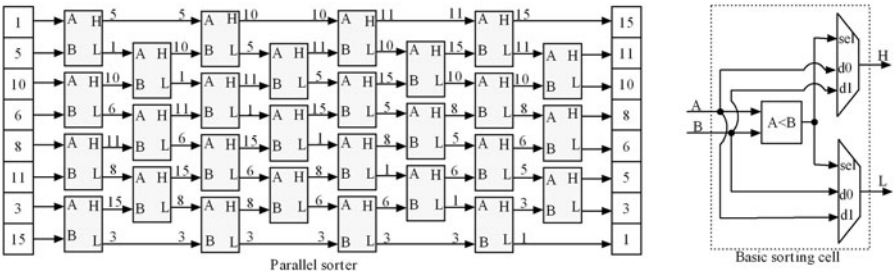


Fig. 8 The *left subfigure* presents the parallel sorting of eight data [14]. The unsorted data values are presented on the *left* register, and the sorted ones are recovered on the *right* register. Each pair of adjacent data values (A and B) are sorted into high (H) and low (L) by the basic block detail in the *right subfigure*

obtained by: (a) sorting the set in ascending order and (b) selecting the value of the middle datum if M is odd or finding the mean of the two middle data if M is even.

As shown in Fig. 7, the threshold estimator block is essentially composed of two sublocks: (a) the absolute value subblock that provides the absolute value of the wavelet coefficients at the first level and (b) the median value subblock that computes the median of the absolute values of the last M wavelet coefficients at the first level. The median value subblock uses a serial input register to keep the last M samples followed by a parallel sorter (Fig. 8) [14], which uses a basic subblock to sort adjacent data values (A and B) into high (H) and low (L). Thus, the threshold is computed by $\lambda = \text{MAD}\sqrt{2\log(N)}/0.6745$, where N is the size of the used signal frame ($N = 2M$ due to the down-sampling by 2).

As for the wavelet analysis/synthesis architecture, the filter path delays of the wavelet shrinkage one are computed as follows:

- Path 1: $\tau = 7(\tau_H + \tau_{\widetilde{H}}) + 8\tau_1$
- Path 2: $\tau = 3(\tau_H + \tau_{\widetilde{H}}) + 4(\tau_G + \tau_{\widetilde{G}}) + 8\tau_2$
- Path 3: $\tau = (\tau_H + \tau_{\widetilde{H}}) + 2(\tau_G + \tau_{\widetilde{G}}) + 4\tau_3$
- Path 4: $\tau = (\tau_G + \tau_{\widetilde{G}}) + 2\tau_4$.

These equations are similar to those obtained in Sect. 2.1.4.

If $\tau_H + \tau_{\widetilde{H}} = \tau_G + \tau_{\widetilde{G}}$, these delays are redefined as:

- Path 1: $\tau = 7(\tau_H + \tau_{\widetilde{H}}) + 8\tau_1$
- Path 2: $\tau = 7(\tau_H + \tau_{\widetilde{H}}) + 8\tau_2$
- Path 3: $\tau = 3(\tau_H + \tau_{\widetilde{H}}) + 4\tau_3$
- Path 4: $\tau = (\tau_H + \tau_{\widetilde{H}}) + 2\tau_4$.

Equalizing Path 1 and Path 2 to their minimum common value, $\tau = 7(\tau_H + \tau_{\widetilde{H}})$, leads to $\tau_1 = \tau_2 = 0$. By equalizing Path 3 and Path 4 to Path 1, one can deduce that $(\tau_3 = \tau_H + \tau_{\widetilde{H}})$ and $\tau_4 = 3(\tau_H + \tau_{\widetilde{H}})$.

2.2.3 Pipelined Real-Time Wavelet-Based Denoising

The pipelined architecture is obtained from the conventional one by pipelining the analysis/synthesis cells, the threshold estimator and the thresholding function without

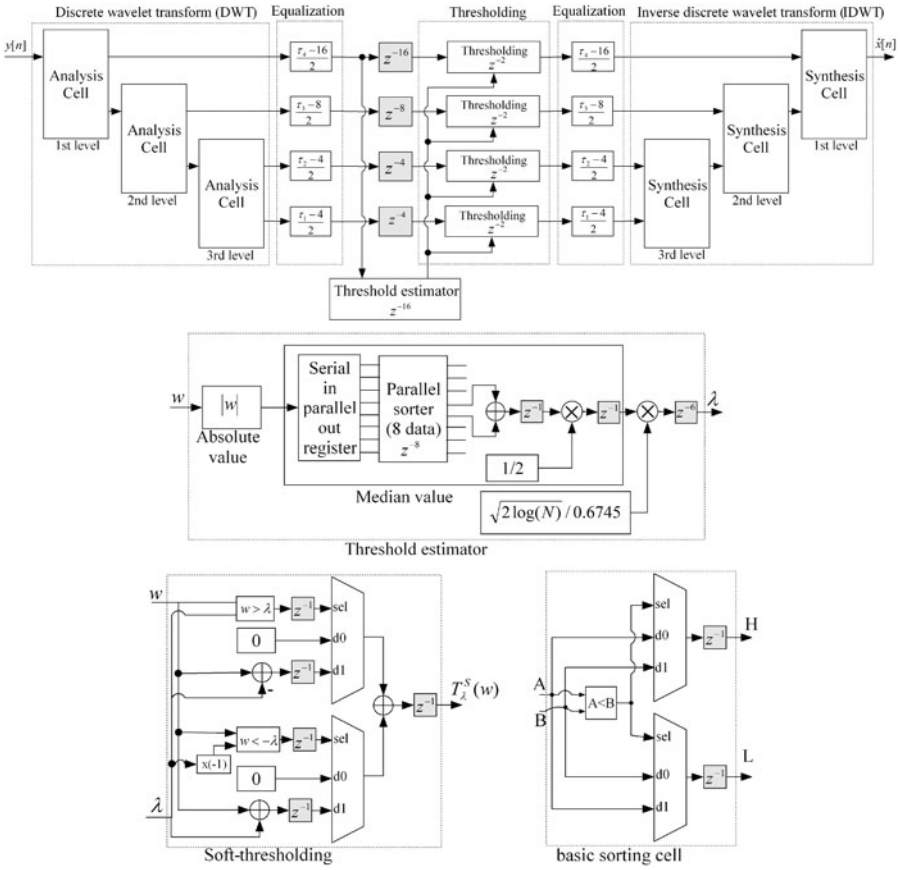


Fig. 9 Diagram of the proposed pipelined real-time architecture of a wavelet-based denoising algorithm. The pipelined threshold estimator and the thresholding blocks are detailed in the *bottom subfigures*

altering the path delays equalization. As shown in Fig. 9, the pipelining leads to a delay of one unit for the basic sorting cell, two units for the soft-thresholding function and sixteen units for the threshold estimator block. The last delay is increased to the next power of 2, i.e., 16. Due to the down-sampling of 2, a delay of 16 samples in Path 4 corresponds to a delay of 8, 4 and 4 samples in Path 3, Path 2 and Path 1, respectively. Inserted delays in different paths are readjusted to take into account delays resulted from pipelining.

For the pipelined architecture of the wavelet denoising using pyramid schema and direct-form FIR filters (D-P-PY-DF), the path delays are obtained as follows:

- Path 1: $\tau = 7(\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 8(\tau_1 + 2)$
- Path 2: $\tau = 3(\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 4(\tau_G + \tau_{\widetilde{G}} + 2N + 1) + 8(\tau_2 + 2)$
- Path 3: $\tau = (\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 2(\tau_G + \tau_{\widetilde{G}} + 2N + 1) + 4(\tau_3 + 2)$
- Path 4: $\tau = (\tau_G + \tau_{\widetilde{G}} + 2N + 1) + 2(\tau_4 + 2)$.

Table 3 Abbreviations used to define the wavelet-based denoising architectures

Abbreviation	Definition
D-C-PY-DF	Wavelet denoising using conventional pyramid algorithm and direct-form FIR
D-C-PY-TF	Wavelet denoising using conventional pyramid algorithm and transposed-form FIR
D-C-PO-DF	Wavelet denoising using conventional polyphase algorithm and direct-form FIR
D-C-PO-TF	Wavelet denoising using conventional polyphase algorithm and transposed-form FIR
D-P-PY-DF	Wavelet denoising using pipelined pyramid algorithm and direct-form FIR
D-P-PY-TF	Wavelet denoising using pipelined pyramid algorithm and transposed-form FIR
D-P-PO-DF	Wavelet denoising using pipelined polyphase algorithm and direct-form FIR
D-P-PO-TF	Wavelet denoising using pipelined polyphase algorithm and transposed-form FIR

These equations are obtained from those of Sect. 2.1.5 by adding the delays corresponding to the thresholding block at different levels.

If $\tau_H + \tau_{\widetilde{H}} = \tau_G + \tau_{\widetilde{G}}$, these delays are redefined as:

- Path 1: $\tau = 7(\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 8(\tau_1 + 2)$
- Path 2: $\tau = 7(\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 8(\tau_2 + 2)$
- Path 3: $\tau = 3(\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 4(\tau_3 + 2)$
- Path 4: $\tau = (\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 2(\tau_4 + 2)$.

Equalizing Path 1 and Path 2 to their minimum common value, $\tau = 7(\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 8(4 + 2)$, leads to $\tau_1 = \tau_2 = 4$. By equalizing Path 3 and Path 4 to Path 1, one can deduce that $\tau_3 = (\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 10$ and $\tau_4 = 3(\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 22$.

For Daubechies 5 wavelet, the filters have 10 coefficients each ($N = 10$). Delays on Path 1 and Path 2 are identical and equal to the overall delay $\tau = 7(9 + 2 \times 10 + 1) + 48 = 258$. By equalizing Path 3 and Path 4 to Path 1, one can obtain $\tau_3 = 40$ and $\tau_4 = 112$.

As for the wavelet analysis/synthesis algorithm, the wavelet-based denoising one can be implemented in different ways, depending on the used schema (pyramid or polyphase), the canonical form of the FIR filters and the pipelining choice. Table 3 presents the abbreviations used to define the eight possible combinations implemented in this work. The filter path delays of these architectures are equalized by inserting appropriate delays. Table 4 presents the inserted delays and the output latency of each architecture. The inserted delays τ_3 and τ_4 of the pipelined architectures are increased compared to those of the conventional ones, due to the delays added during the pipelining process. These delays are more important with the direct form than with the transposed form. Also, it can be seen that the conventional architectures present the same output latency of 63 samples. The output latencies of the pipelined architectures are more important than those of the conventional one, where the largest value of 286 samples is obtained for AS-P-PO-DF architecture.

Table 4 Inserted delay values, expressed in samples, for eight wavelet-based denoising architectures. The output latency is evaluated for 3-level schema using Daubechies 5 wavelet, i.e., $\tau_H + \tau_{\widetilde{H}} = 9$ and $N = 10$

Architecture	τ_1	τ_2	τ_3	τ_4	τ	Latency
D-C-PY-DF	0	0	$\tau_H + \tau_{\widetilde{H}}$	$3(\tau_H + \tau_{\widetilde{H}})$	$7(\tau_H + \tau_{\widetilde{H}})$	63
D-C-PY-TF	0	0	$\tau_H + \tau_{\widetilde{H}}$	$3(\tau_H + \tau_{\widetilde{H}})$	$7(\tau_H + \tau_{\widetilde{H}})$	63
D-C-PO-DF	0	0	$\tau_H + \tau_{\widetilde{H}}$	$3(\tau_H + \tau_{\widetilde{H}})$	$7(\tau_H + \tau_{\widetilde{H}})$	63
D-C-PO-TF	0	0	$\tau_H + \tau_{\widetilde{H}}$	$3(\tau_H + \tau_{\widetilde{H}})$	$7(\tau_H + \tau_{\widetilde{H}})$	63
D-P-PY-DF	4	4	$\tau_H + \tau_{\widetilde{H}} + 2N + 1 + 10$	$3(\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 22$	$7(\tau_H + \tau_{\widetilde{H}} + 2N + 1) + 48$	258
D-P-PY-TF	4	4	$\tau_H + \tau_{\widetilde{H}} + 5 + 10$	$3(\tau_H + \tau_{\widetilde{H}} + 5) + 22$	$7(\tau_H + \tau_{\widetilde{H}} + 5) + 48$	146
D-P-PO-DF	4	4	$\tau_H + \tau_{\widetilde{H}} + 2(N + 2) + 1 + 10$	$3(\tau_H + \tau_{\widetilde{H}} + 2(N + 2) + 1) + 22$	$7(\tau_H + \tau_{\widetilde{H}} + 2(N + 2) + 1) + 48$	286
D-P-PO-TF	4	4	$\tau_H + \tau_{\widetilde{H}} + 13 + 10$	$3(\tau_H + \tau_{\widetilde{H}} + 13) + 22$	$7(\tau_H + \tau_{\widetilde{H}} + 13) + 48$	202

3 FPGA Implementation

The proposed architectures were implemented on FPGA using Xilinx System Generator (XSG) and XUP Virtex-II Pro Development System. XSG is a high-level software tool that enables the use of MATLAB/Simulink environment to create and verify hardware designs for Xilinx FPGAs quickly and easily. It provides a library of Simulink blocks bit and cycle accurate modeling for arithmetic and logic functions, memories and DSP functions. It also includes a code generator that automatically generates HDL code from the created model. Generated HDL code can be synthesized and implemented in the Xilinx FPGAs. The XSG blocks are like standard Simulink blocks except that they can operate only in discrete-time and fixed-point format [4].

3.1 Wavelet Transform

The eight wavelet analysis/synthesis architectures defined in Table 1 are implemented on FPGA. Figures 10 and 11 present the conventional and the pipelined wavelet analysis/synthesis architectures, respectively. Each module is identified by the label on the block, where Ana_Cell and Syn_Cell correspond to the analysis cell and the synthesis cell, respectively, and LO_D, HI_D, LO_R and HI_R correspond to H, G, $\widetilde{H}$ and $\widetilde{G}$, respectively. It can be noted that the inserted delays and the output latency correspond to the values listed in Table 2.

Table 5 gives the resources required by the proposed architectures and the maximum operating frequency when the fixed-point data are represented by 2's complement signed 18-bit number. For the conventional architectures, it can be seen that the direct and transposed forms use exactly the same number of resources either for pyramid or polyphase schemas. Compared to pyramid schema, the polyphase one reduces the number of multiplications (see Sect. 2.1.2) but not the number of multipliers. Thus, the maximum operating frequency of the AS-C-PO-DF and AS-C-PO-TF are greater than those of AS-C-PY-DF and AS-C-PY-TF ones, respectively, because the transposed-form FIR presents a smaller critical path than the direct-form FIR.

As expected, the pipelined architectures require more resources (Slices, Flip-flops and 4-LUTs) than the conventional corresponding ones. Also, the AS-P-PY-DF and AS-P-PO-DF require more resources than AS-P-PY-TF and AS-P-PO-TF, respectively, because the pipelining of the direct form needs more delays than that of the transposed one. For the pipelined architectures, the maximum operating frequency is approximately constant because the critical path is reduced to a single multiplication operation.

3.2 Wavelet Shrinkage

The eight wavelet-based denoising architectures defined in Table 3 are implemented on FPGA. Figures 12 and 13 present the conventional and the pipelined wavelet-based denoising architectures, respectively. These figures present also the conventional and the pipelined circuits for the soft-thresholding function and the threshold estimator. It can be noted that the inserted delays and the output latency correspond to the values listed in Table 4.

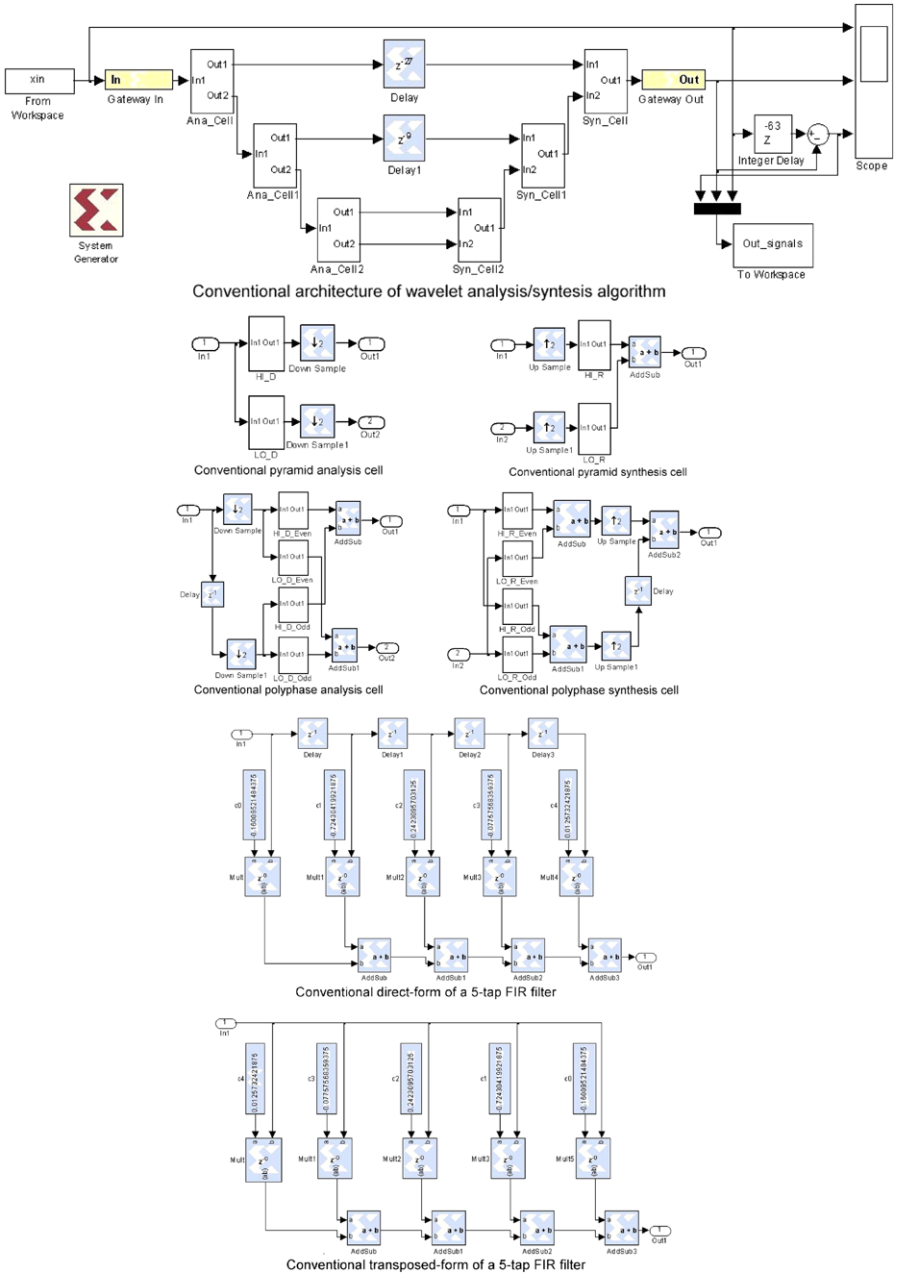


Fig. 10 Implementation of the conventional wavelet analysis/synthesis algorithms. Architectures of the analysis/synthesis cells and the canonical forms of the wavelet-based FIR filters are also detailed

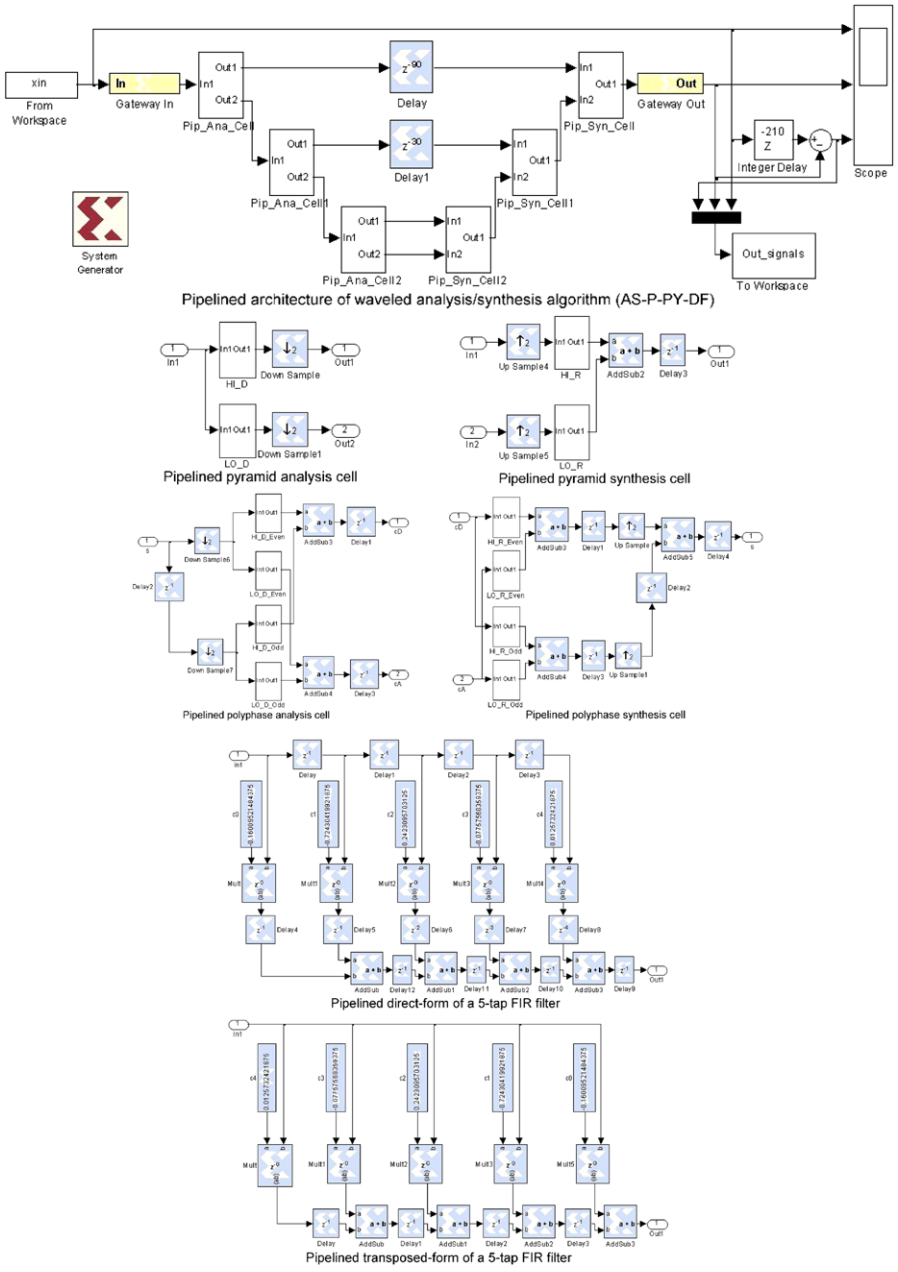


Fig. 11 As in Fig. 10 but with the pipelined wavelet analysis/synthesis algorithms

Table 5 Resource utilization and maximum operating frequency for conventional and pipelined wavelet analysis/synthesis architectures. Resource availability of Xilinx Virtex-II Pro XC2VP30 FPGA are given between brackets

Architecture	Conventional architectures				Pipelined architectures			
	AS-C-PY-DF	AS-C-PY-TF	AS-C-PO-DF	AS-C-PO-TF	AS-P-PY-DF	AS-P-PY-TF	AS-P-PO-DF	AS-P-PO-TF
Resource utilization								
Slices (13,696)	1,239	1,239	1,185	1,185	3,569	2,787	3,547	2,975
Flip-flops (27,392)	2,477	2,477	2,369	2,369	6,725	4,925	6,617	5,285
4-LUTs (27,392)	2,240	2,240	2,273	2,273	4,058	2,186	3,659	2,327
18-bit MULT (136)	120	120	120	120	120	120	120	120
GClocks (16)	1	1	1	1	1	1	1	1
Bonded IOBs (556)	37	37	37	37	37	37	37	37
Equivalent gate	525,003	525,003	524,151	524,151	675,387	545,763	647,097	551,163
Max. Ope. Freq. (MHz)	9.759	25.222	12.168	18.790	163.470	158.173	163.470	160.899

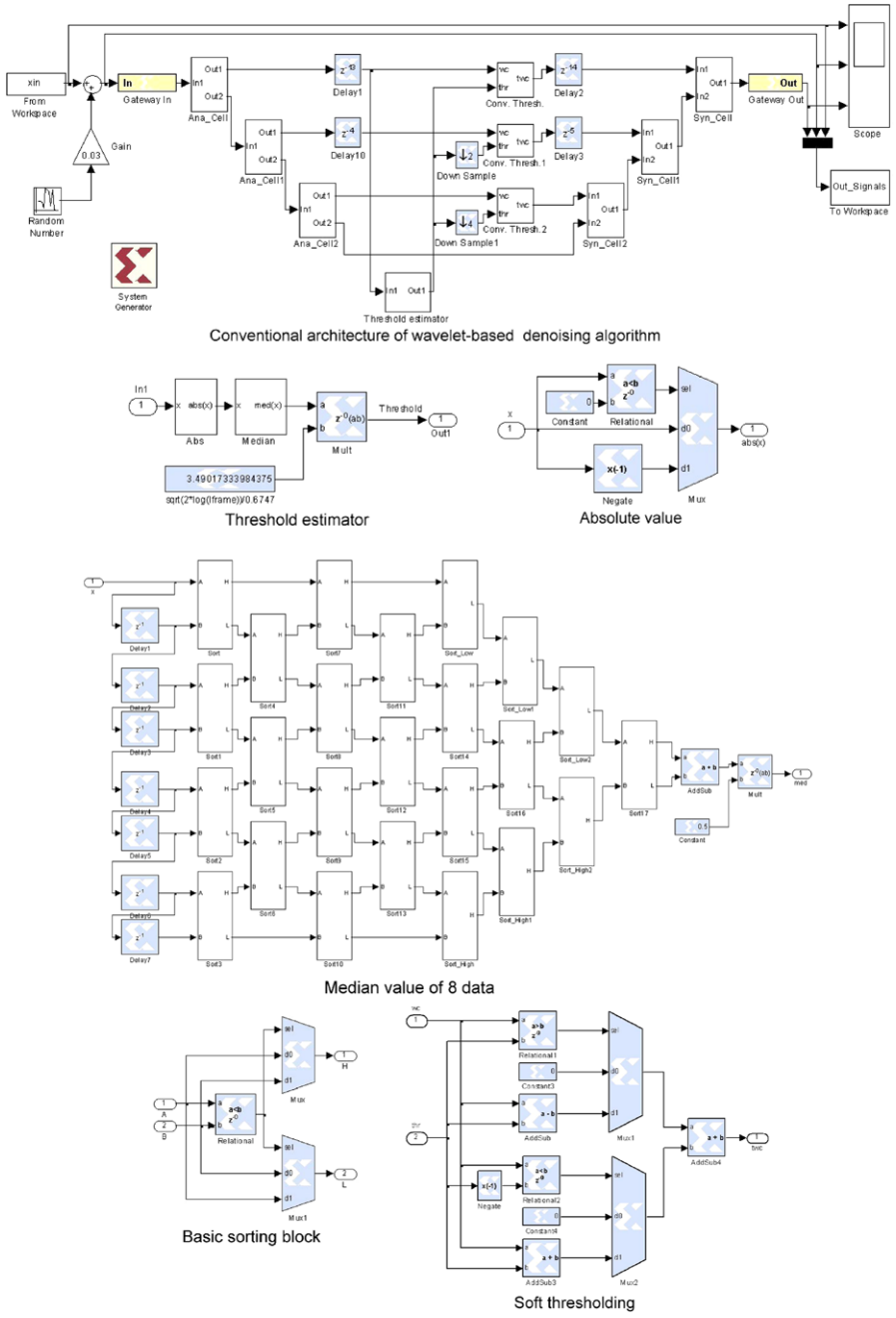
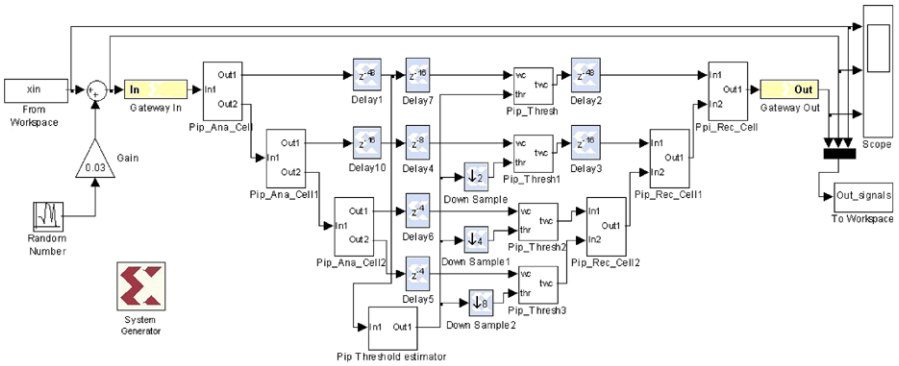
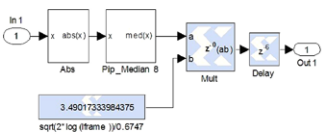


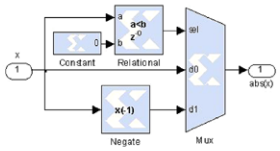
Fig. 12 Implementation of the conventional wavelet-based denoising algorithms. Architectures of the soft-thresholding function, the threshold estimator and the median value computation are also detailed



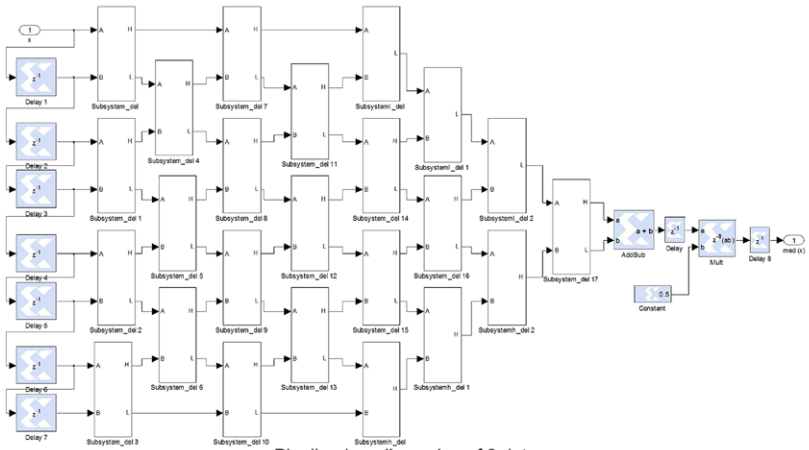
Pipelined architecture of waveled-based denoising algorithm (D-P-PY-DF)



Pipelined threshold estimator



Absolute value



Pipelined median value of 8 data

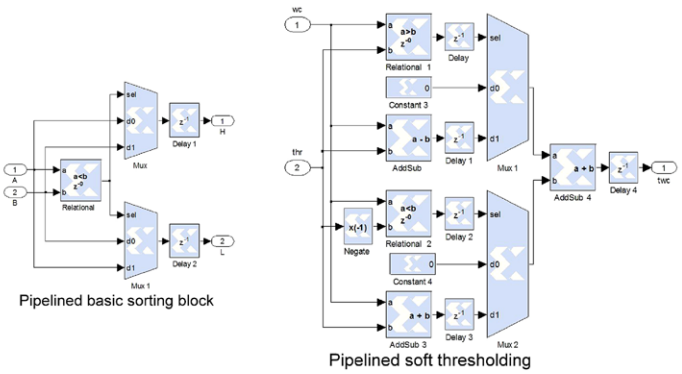


Fig. 13 As in Fig. 12 but with the pipelined wavelet-based denoising algorithms

Table 6 Resource utilization and maximum operating frequency for conventional and pipelined for wavelet-based denoising architectures. Resource availability of Xilinx Virtex-II Pro XC2VP30 FPGA are given between brackets

Architecture	Conventional architectures				Pipelined architectures			
	D-C-PY-DF	D-C-PY-TF	D-C-PO-DF	D-C-PO-TF	D-P-PY-DF	D-P-PY-TF	D-P-PO-DF	AS-P-PO-TF
Resource utilization								
Slices (13,696)	3,473	2,608	3,394	2,647	4,787	4,020	4,762	4,191
Flip-flops (27,392)	2,685	2,685	2,577	2,577	8,135	6,371	8,027	6,695
4-LUTs (27,392)	4,190	4,190	4,187	4,187	6,178	4,414	5,743	4,411
18-bit MULT (136)	122	122	122	122	122	122	122	122
GClocks (16)	1	1	1	1	1	1	1	1
Bonded IOBs (556)	37	37	37	37	37	37	37	37
Equivalent gate	550,573	550,573	549,493	549,493	717,748	590,710	689,082	593,148
Max. Ope. Freq. (MHz)	9.249	14.653	11.478	13.845	160.363	158.173	160.363	160.363

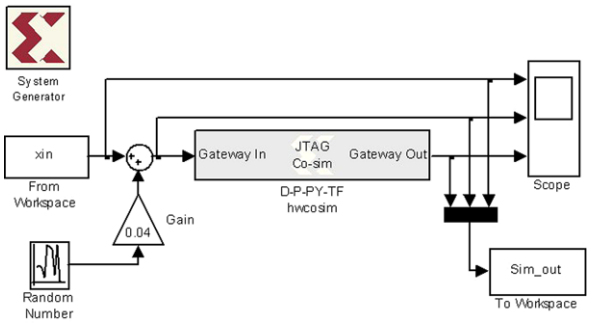


Fig. 14 Hardware-in-the-loop co-simulation of pipelined wavelet-based denoising algorithm

Table 6 gives the resources required by the proposed architectures when the fixed-point data are represented by 2's complement signed 18-bit number. For the conventional architectures, it can be seen that the direct and transposed forms use approximately the same number of resources either for pyramid or polyphase schemas. As for the conventional analysis/synthesis architectures, the maximum operating frequencies of the D-C-PO-DF and D-C-PO-TF are greater than those of D-C-PY-DF and D-C-PY-TF ones, respectively. This can also be explained by the smallest critical path of the transposed-form FIR.

As previously, the pipelined architectures require more resources (Slices, Flip-flops and 4-LUTs) than the conventional corresponding ones. Also, the D-P-PY-DF and D-P-PO-DF require more resources than D-P-PY-TF and D-P-PO-TF, respectively, because the pipelining of the direct form needs more delays than that of the transposed one. This table shows that the maximum operating frequency is approximately constant for the pipelined architectures because the critical path is reduced to a single multiplication operation.

Tables 5 and 6 show that the hardware implementation of the proposed architectures is mainly limited by the number of multipliers available on the used FPGA. In fact, the universal thresholding step consumes only two additional multipliers (122 instead of 120). Therefore, parallel implementation of advanced wavelet-based denoising algorithm that needs more multipliers will require large FPGAs.

After successful simulation, the hardware co-simulation compilation automatically creates bitstream file and associates it with a JTAG co-simulation block (shaded block in Fig. 14). The hardware-in-the-loop co-simulation enables incorporating the design running in an FPGA directly into a Simulink simulation. When the design is simulated, the compiled portion (JTAG co-simulation block) is actually running on the hardware, and data is transferred between computer and FPGA board [4].

In fact, the hardware-in-the-loop co-simulation can be accomplished only for the pipelined architectures because the maximum of their operating frequencies (Tables 5 and 6) is greater than the system clock frequency (100 MHz) of the used board. Figure 14 shows the diagram of the hardware-in-the-loop co-simulation that enables running the compiled model (wavelet-based denoising) on the FPGA.

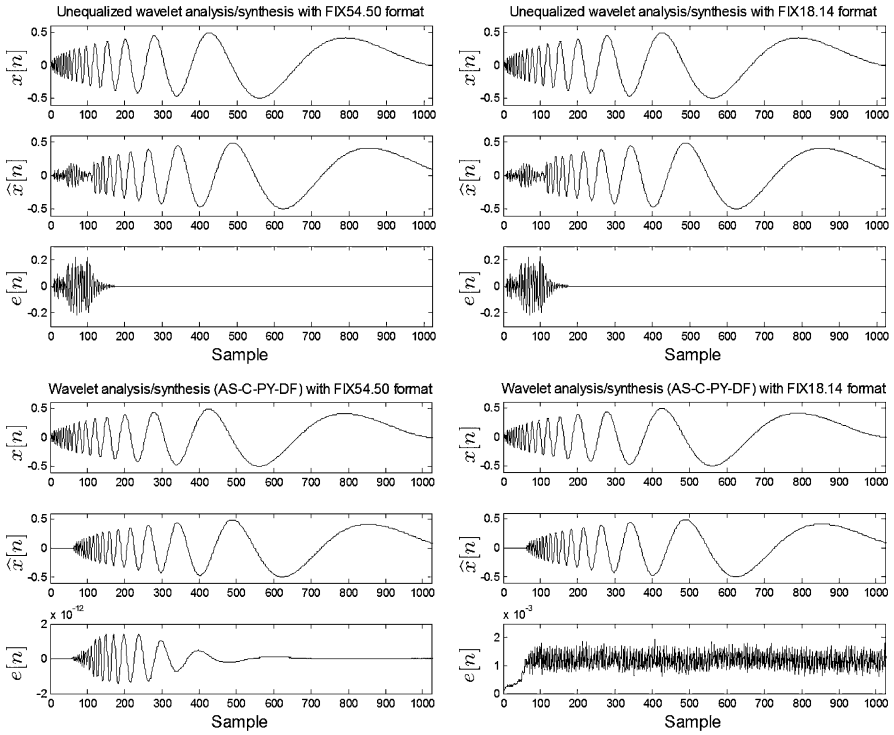


Fig. 15 Results of simulation obtained by a conventional pyramid analysis/synthesis architecture without (*top*) and with (*bottom*) path delays equalization for various fixed-point formats. For example, the fixed-point format FIX18.14 is a 2's complement signed 18-bit number having 14 fractional bits. For each *subfigure*, the original signal $x[n]$ is given on top, followed by the reconstructed signal with equalization $\widehat{x}[n]$ and the error signal $e[n]$ that is defined by $x(n-\tau)-\widehat{x}[n]$

4 Results and Discussion

Artificial signals are used to evaluate the proposed architectures as follows. Firstly, the wavelet analysis/synthesis architectures are evaluated in term of the reconstruction error. Secondly, the wavelet-based denoising architecture are evaluated in term of similarity between the original signal and the cleaned one.

4.1 Wavelet Transform

Figure 15 presents the results obtained by simulation using the conventional pyramid architecture (AS-C-PY-DF) for various fixed-point formats. Characterized by filter delays equalization, this architecture is compared with similar one but without filter delays equalization. It can be shown that architecture with equalization guarantees a perfect reconstitution, $|e[n]| < 1.5 \times 10^{-12}$, when quantization uses more than 54 bits. Near perfect reconstruction, $|e[n]| < 2 \times 10^{-3}$, is obtained using only 18 bits. However, architecture without equalization, presents an important error regardless of the number of bits. This can be explained by the wavelet coefficient desynchronization through analysis and synthesis paths. It can be noted that the delay of

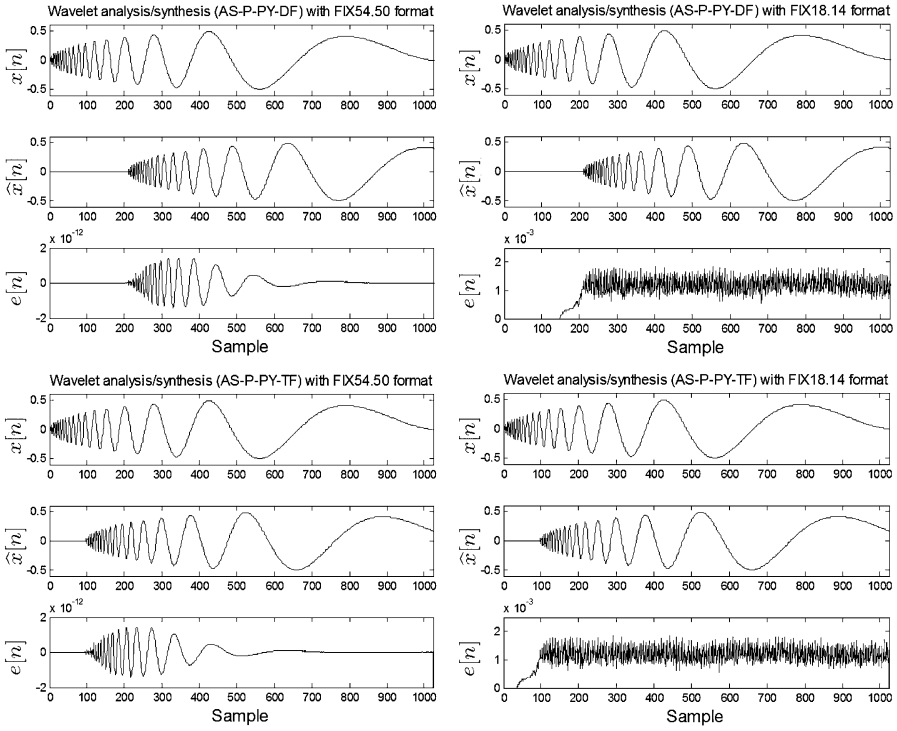


Fig. 16 As in Fig. 15 but for results of hardware-in-the-loop co-simulation obtained with the pipelined pyramid architectures

63 samples between the original signal $x[n]$ and the reconstructed one $\widehat{x}[n]$ corresponds to the output latency given in Table 2. The reconstruction error is defined by $e[n] = x[n-\tau] - \widehat{x}[n]$. Reconstruction errors obtained by the other three conventional wavelet analysis/synthesis architectures (AS-C-PY-TF, AS-C-PO-DF and AS-C-PO-TF) are similar to that obtained by AS-C-PY-DF (Fig. 15). The four conventional analysis/synthesis wavelet transform architectures (AS-C-PY-DF, AS-C-PY-TF, AS-C-PO-DF and AS-C-PO-TF) present also the same output delay of 63 samples, according to Table 2.

On the other hand, Fig. 16 presents the results obtained by hardware co-simulation using pipelined pyramid architectures (AS-P-PY-DF and AS-P-PYO-DF). These architectures give the same construction errors as those obtained by the conventional ones, but they present higher output latencies. In fact, the pipelining approach inserts delays in various paths (to reduce the critical path and consequently increase the operating frequency), but it preserves the wavelet coefficient synchronization. The AS-P-PY-DF presents higher output delay (210 samples) than that obtained with AS-P-PY-TF (98 samples). The output delay differences can be explained by the fact that AS-P-PY-DF uses pipelined direct form FIR filters that present larger output delay than pipelined transposed form FIR filters used in AS-P-PY-TF (see Fig. 5 for more details). The two other pipelined architectures (AS-P-PY-DF and AS-P-PO-DF) present the same reconstruction errors as those obtained by AS-P-PO-DF and AS-P-

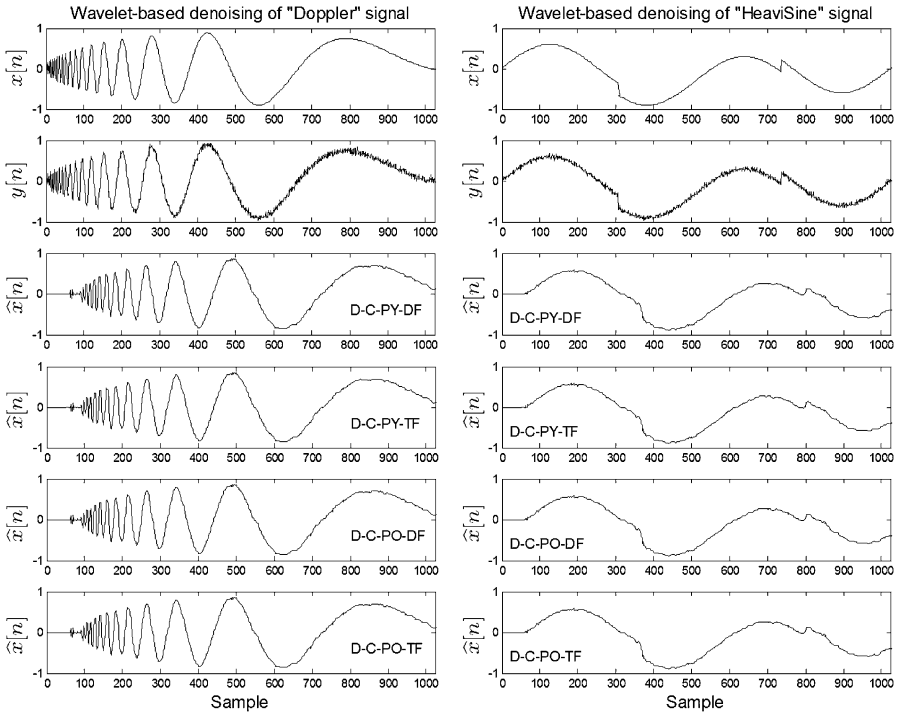


Fig. 17 Results of simulation obtained with the conventional wavelet-based denoising architecture using two synthetic signals. The original signal $x[n]$ is given on top, followed by the noised version $y[n]$ and the filtered output $\widehat{x}[n]$ for various conventional denoising architectures

PO-DF with error latencies of 238 and 154 samples, respectively. The polyphase pipelined architectures present slightly higher output latencies than the pyramid ones because they have longest critical path (additional adders as can be seen in Fig. 1).

Finally, the proposed equalized path architectures presents negligible reconstruction error compared to the unequalized ones. However, they present output latencies that are increased by pipelining. The lowest output latency (98 samples) of the pipelined architectures is obtained with AS-P-PY-TF, which is based on the pyramid schema and the transposed-form FIR filters.

4.2 Wavelet Shrinkage

Figure 17 shows the denoising results obtained by simulation using the conventional architectures (D-C-PY-DF, D-C-PY-TF, D-C-PO-DF and D-C-PO-TF) and two artificial signals. The noise is efficiently removed without altering the original signals. However, the filtered outputs ($\widehat{x}[n]$) present delays, which are introduced by the filter delays equalization. The four architectures are characterized by the same output delay of 63 samples, which corresponds to the output latencies listed in Table 4.

On the other hand, Fig. 18 presents the results obtained by hardware co-simulation using pipelined architectures (D-P-PY-DF, D-C-PY-TF, D-P-PO-DF and D-P-PO-

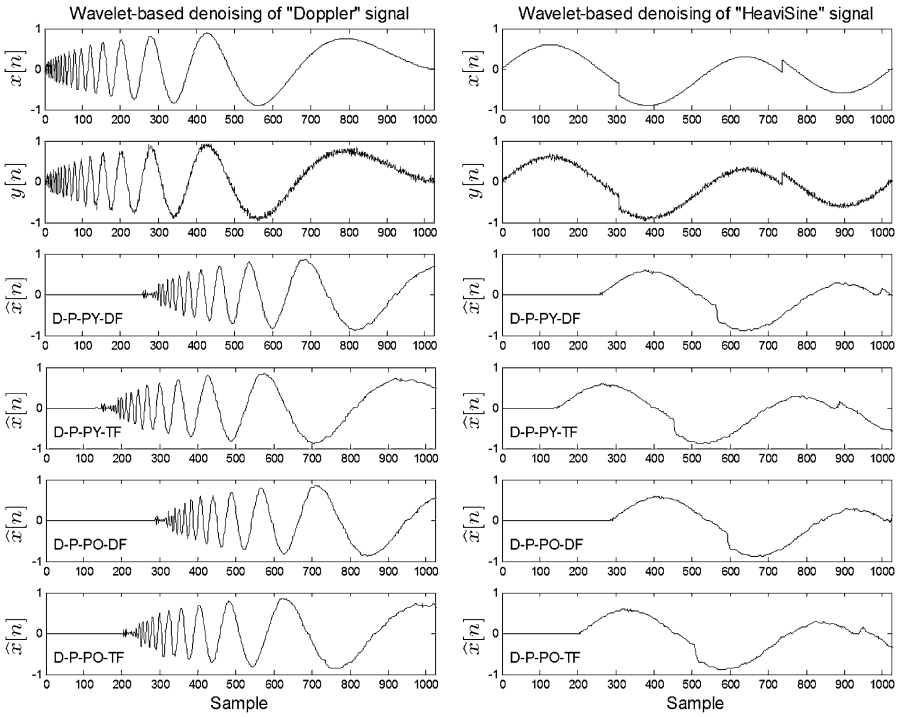


Fig. 18 As in Fig. 17 but with results of hardware-in-the-loop co-simulation obtained the pipelined denoising architectures

TF). They show the same filtering performances as those obtained with the conventional ones, but they present higher output latencies. As previously, the pipelining approach inserts delays in various filter paths, but it preserves the wavelet coefficient synchronization.

Finally, the pipelined wavelet-based denoising architectures present the same performances as those obtained by the conventional ones, except the output latencies that are increased by pipelining. For the pipelined architectures, the lowest output latency (146 samples) is obtained by the D-P-PY-TF architecture, which is based on the pyramid schema and the transposed-form FIR filters.

5 Conclusion

A real-time wavelet transforms and their applications to signal denoising are implemented on FPGA. It is shown that in addition to the wavelet orthogonality, the filter path delays must be equalized in the forward/inverse wavelet transforms to ensure perfect reconstruction. Near-perfect reconstruction is obtained using a fixed-point data of only 18 bits. The universal threshold estimated on few samples allows one to remove moderate noise efficiently and in real-time without altering the original signal.

In the future, more advanced methods for estimating the noise level will be implemented and tested on real signals as electrocardiogram (ECG) and speech.

Acknowledgement The authors thank Professor Jean Rouat of Université de Sherbrooke for proof reading.

References

1. K. Andra, C. Chakrabarti, T. Acharya, A VLSI architecture for lifting-based forward and inverse wavelet transform. IEEE Trans. Signal Process. **50**(4), 966–977 (2002)
2. K. Azadet, C.J. Nicole, Low-power equalizer architectures for high-speed modems. IEEE Commun. Mag. **36**(10), 118–126 (1998)
3. M. Bahoura, H. Ezzaidi, Real-time implementation of discrete wavelet transform on FPGA, in *IEEE 10th International Conference on Signal Processing (ICSP)*, Oct. (2010), pp. 191–194
4. M. Bahoura, H. Ezzaidi, FPGA—implementation of parallel and sequential architectures for adaptive noise cancelation. Circ. Syst. Signal Process. 1–28. doi:10.1007/s00034-011-9310-0
5. M. Bahoura, J. Rouat, Wavelet speech enhancement using the teager energy operator. IEEE Signal Process. Lett. **8**, 10–12 (2001)
6. M. Bahoura, J. Rouat, Wavelet speech enhancement based on time-scale adaptation. Speech Commun. **48**(12), 1620–1637 (2006)
7. S.G. Chang, B. Yu, M. Vetterli, Adaptive wavelet thresholding for image denoising and compression. IEEE Trans. Image Process. **9**(9), 1532–1546 (2000)
8. J. Chilo, T. Lindblad, Hardware implementation of 1D wavelet transform on an FPGA for infrasound signal classification. IEEE Trans. Nucl. Sci. **55**(1), 9–13 (2008)
9. D.L. Donoho, Nonlinear wavelet methods for recovering signals, images, and densities from indirect and noisy data. Proc. Symp. Appl. Math. **47**, 173–205 (1993)
10. D.L. Donoho, De-noising by soft-thresholding. IEEE Trans. Inf. Theory **41**, 613–627 (1995)
11. A. Grzeszczak, M.K. Mandal, S. Panchanathan, VLSI implementation of discrete wavelet transform. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **4**(4), 421–433 (1996)
12. K.A. Kotteri, S. Barua, A.E. Bell, J.E. Carletta, A comparison of hardware implementations of the biorthogonal 9/7 DWT: convolution versus lifting. IEEE Trans. Circuits Syst. II, Express Briefs **52**(5), 256–260 (2005)
13. S. Mallat, A theory for multiresolution signal decomposition: the wavelet representation. IEEE Trans. Pattern Anal. Mach. Intell. **11**, 674–693 (1989)
14. J. Martinez, R. Cumplido, C. Feregrino, An FPGA-based parallel sorting architecture for the Burrows Wheeler transform, in *Proceedings of the 2005 International Conference on Reconfigurable Computing and FPGAs*, (2005) 7 pp.
15. Matlab, *Signal Processing Blockset 7 User's Guide* (The MathWorks, Inc., Natick, 2010)
16. K.G. Oweiss, A. Mason, Y. Suhail, A.M. Kamboh, K.E. Thomson, A scalable wavelet transform VLSI architecture for real-time signal processing in high-density intra-cortical implants. IEEE Trans. Circuits Syst. **54**(6), 1266–1278 (2007)
17. K.K. Parhi, T. Nishitani, VLSI architectures for discrete wavelet transforms. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **1**(2), 191–202 (1993)
18. S. Poornachandra, Wavelet-based denoising using subband dependent threshold for ECG signals. Digit. Signal Process. **18**(1), 49–55 (2008)
19. R. Quian Quiroga, Obtaining single stimulus evoked potentials with wavelet denoising. Physica D: Nonlinear Phenom. **145**(3–4), 278–292 (2000)
20. L. Su, G. Zhao, De-Noising of ECG signal using translation-invariant wavelet De-Noising method with improved thresholding, in *27th Annual International Conference of the IEEE EMBS*, Sept. (2005), pp. 5946–5949
21. P.E. Tikkanen, Nonlinear wavelet and wavelet packet denoising of electrocardiogram signal. Biol. Cybern. **80**(4), 259–267 (1999)
22. A. Vera, U. Meyer-Baese, M. Pattichis, An FPGA based rapid prototyping platform for wavelet coprocessors, in *Proceedings of SPIE—The International Society for Optical Engineering*, vol. 6576, pp. 657615.1–657615.10 (2007)
23. C. Wang, W.S. Gan, Efficient VLSI architecture for lifting-based discrete wavelet packet transform. IEEE Trans. Circuits Syst. II **54**(5), 422–426 (2007)