New Architecture of Wavelet Packet Transform Using Parallel Filters

HOSSEIN AMIRI FARAHANI, MAJID MASHAYEKHI NEZAMABADI
Department of Electrical engineering faculty,
Pakdasht Sama College of Azad University
TEHRAN, IRAN
hossein.amiri1974@gmail.com, majid.mashayekhi1967@gmail.com

Abstract-In this paper, based on the word-serial pipeline architecture and parallel filter processing, a new architecture for direct and inverse wavelet packet transforms is introduced. This architecture increases the speed of the wavelet packet transforms. In this design a word–serial architecture able to compute a complete wavelet packet transform (WPT) binary tree in an on-line fashion, but easily configurable in order to compute any required WPT sub tree, is proposed. In this architecture, a high-pass filter and a low-pass filter are used concurrently, in order to compute the new coefficients. This architecture is suitable for the high-speed on-line applications. With this architecture, the speed of the wavelet packet transforms is increased with a factor two, but the occupied area of the circuit is less than double. This architecture can be applied to any levels tree structure with any filter coefficients length.

Keywords: WPT, tree structure, parallel filters

1. Introduction
Wavelets, based on the time-scale representations, provide an alternative to the time-frequency representation based signal processing. The shifting (or translation) and the scaling (or dilation) are unique to wavelets. The Wavelet is a kind of bases which are generated by dilation and translation of a function, [2] and [3]. The wavelet analysis method has a good ability at localizing a signal in both time and frequency plane [4].

Wavelet transform has many applications such as: signal and image compression, speech and video coding [5], Multimode transmission [6] and denoising [7]. The performance of most of these applications increases when the transform provides good spectral and temporal resolution in arbitrary region of the time-frequency plane. Wavelet packet transform (WPT) provided this resolution.

There are several architectures for WPT. For example in [8] Vishwanath and Owens introduce a common architecture for the DWT and IDWT. Bing-Fei Wu and Yi-Qiang Hu Introduce an Efficient architecture of the Discrete Wavelet Transform Using Embedded Instruction Codes for Symmetric Filters by two multipliers and four adders [9]. Tracy C. Denk and Keshab K. Parhi describe an orthonormal DWT architecture which uses the QMF lattice structure and digit-serial processing techniques [10].

In this paper, a new architecture for the wavelet packet transforms is introduced. The algorithm for tree structure of wavelet packet transforms is analyzed in the section two. The direct Wavelet Packet Transform architecture is described in the section three. This section contains the memories and processors structure, data flow in the processor and the data dependency. The inverse architecture of wavelet packet transform is explained briefly in the section four. In the last sections, the conclusion and references are brought.

2. Direct Wavelet Packet Transform Algorithm

The model used in [4] to implement the tree structure of Direct Wavelet Packet Transform (DWPT) is based on the filtering process. Figure 1 depicted a complete 3-level Direct WPT. In this Figure G and H are the high-pass and low-pass half-band filters, respectively.

Figure1. Complete 3-levels for WPT.
Computation period is the number of input cycles for one time produces output samples. In general, the computation period is \( M = 2^j \) for a J-level DWPT. The period of the 3-level computation is 8.

At each level, the coefficients, \( y_{i,j} \)'s, are calculated using the values of coefficients, \( y_{i,j} \)'s, in the previous level, Equations (1) and (2) show relation among the values of the coefficients in this architecture.

On the basis of notations in Figure 1, the samples of a node \((i,j)\) in the tree, are obtained from the output samples of its parent-node \((i-I, j/2)\).

\[
y_{i,j}(n) = e_{i,j}y_{i-1,j/2}(n-2^{i-1}k) \text{ (For high-pass filter)}
\]

(1)

\[
y_{i,j}(n) = e_{i,j}y_{i-1,j/2}(n-2^i-1k) \text{ (For low-pass filter)}
\]

(2)

Where \( i,j \in Z \), \( 0 \leq i < J \), \( 0 \leq j < 2^i \), \( n = 2^i l \), \( l \in Z \) and \( y_{0,0} \) are the input samples. \( j \) is odd for the high-pass filters and \( j \) is even for the low-pass filters. \( h(k) \) and \( g(k) \) are coefficients of the low-pass and the high-pass non-recursive FIR digital filters, respectively.

### 3. Direct Wavelet Packet Transform architecture

According to the Figure 1 and Equations (1) and (2), number of samples is reduced to half after each filtering. The filters only compute half of the samples when moving from one resolution level to the next. But the number of filters increases by a factor of two. The number of coefficients which are computed in each level in the computation-period is \( M \). The total number of coefficients is computed in a DWPT with \( J \) levels is \( J \times M \). Therefore, the number of samples to be computed at each sampling-period is \( \frac{2^J M}{2^J} - J \). Based on the number of computed samples in each level and using 2 parallel filters for using 100% of hardware the number of filters must be twice the number of levels, that’s 2\( J \). Each processor has 2 filters therefore J-level decomposition needs \( J \) processors (see Figure 2, 3). In this paper, processors are considered 3, therefore \( J = 3 \).

The main Architecture for 3-level DWPT is depicted in the Figure 2. In this Figure \( P_i \) and \( M_i \) are the processor and the memory in the \( i \)'th level, for \( 0 \leq i \leq J-1 \).

![Figure 2. Architecture for 3 levels DWPT](image)

Each level contains a processor and a memory. \( P_i \) produces samples for the \((i+1)\)'th level. \( M_i \) stores the intermediate coefficients for \( P_i \) to produce \( 2^{i+1} \) bands in the \((i+1)\)'th level.

The detail architecture of the \( i \)'th level is depicted in Figure 3. Any processor has 2 processing parts, containing a high-pass filter and a low-pass filter. These filters work concurrently.

The structure of each memory is explained in the section 3.1 and the structure of each processor is explained in the section 3.2.

![Figure 3- architecture for one level decomposition](image)

### 3.1 Memory Structure

As shown in Figure 2 the memory of each level (\( M_i \)) has \( 2^i \times L \) logical cells, where \( L \) is the number of filter coefficients and considered 4 in this paper.

\( M_0 \) has four logical cells and stores four samples of the input signal, \( y_{0,0} \). \( y_{0,0} \) is the input sample of the system. The four samples of \( y_{0,0} \) are stored at \( M_0(k) \) for \( k = 0,1,2,3 \).

\( M_1 \) has eight logical cells. The first four logical cells store the four coefficients of the \( y_{1,1} \), this part of \( M_1 \) is called \( M_{1,1} \). \( y_{1,1} \) is the output sample of the high-pass filter of the level 0. The four samples of \( y_{1,1} \) are stored at \( M_{1,1}(k) \) for \( k = 0,1,2,3 \). The second four logical cells store four coefficients of the \( y_{1,0} \), this part of \( M_1 \) is called \( M_{1,0} \). \( y_{1,0} \) is the output sample of the low-pass filter of the level 0. The four samples of \( y_{1,0} \) are stored at \( M_{1,0}(k) \) for \( k = 0,1,2,3 \).

\( M_2 \) has sixteen logical cells. The first eight logical cells store the four coefficients of the \( y_{2,2} \) and four coefficients of the \( y_{2,1} \) respectively. This part of \( M_2 \) is called \( M_{2,2} \). \( y_{2,1} \)'s and \( y_{2,2} \)'s are the output samples of the high-pass filter of the level 1. The eight samples of \( y_{2,2} \) and \( y_{2,1} \) are stored at \( M_{2,2}(k) \) for \( k = 0,1, \ldots, 7 \). The second eight logical cells store the four coefficients of \( y_{2,3} \) and the four coefficients of \( y_{2,0} \) respectively. This part of \( M_2 \) is called \( M_{2,0} \). \( y_{2,0} \)'s and \( y_{2,2} \)'s are the output samples of the low-pass filter of the level 1. The eight samples of \( y_{2,2} \) and \( y_{2,0} \) are stored at \( M_{2,0}(k) \) for \( k = 0,1, \ldots, 7 \).

Each processor at each level produces two coefficients at the same time. Therefore the memory at levels 1 and 2 are divided into 2 separate parts to be written concurrently.

### 3.2 Processor Structure

The structure of each processor is depicted in Figure 3, containing two filters: one high-pass filter and one low-pass filter. The inputs of each processor come from the previous memory unit, and the output
of each processor is fed into the next memory, except $P_2$. The outputs of the processor of level 2, $P_2$, are the output of the DWPT structure.

As shown in Figure 3 the inputs of each processor, $y_{i,j}$, are applied to the high-pass and the low-pass filters simultaneously. Because the delays of the filters are equal, the outputs of the filters are also produced simultaneously.

Calculation of each output requires $L$ clock cycles. The inputs of each filter are multiplied to the filter coefficients and are accumulated in the internal accumulator, ACC. In the L’s clock cycle the result is stored in the next memory.

$P_0$ has the inputs from memory $M_0$. The outputs are fed into the memory $M_1$, Figure 3. The outputs of the high-pass filter of this processor are stored at the first part of the memory of the level 1, $M_{1,1}$. The outputs of the low-pass filter of this processor are stored at the second part of the memory of the level 1, $M_{1,0}$.

$P_1$ has the inputs from memory $M_1$. The outputs are fed into the memory $M_2$, Figure 3. The outputs of the high-pass filter of this processor are stored at the first part of the memory of the level 1, $M_{2,1}$, and the outputs of the low-pass filter of this processor are stored at the second part of the memory of the level 1, $M_{2,0}$.

$P_2$ has the inputs from memory $M_2$. The outputs are fed into the memory $M_3$. The outputs of this processor are the outputs of the DWPT structure.

### 3.3 Dataflow in processors

Each processor calculates the new coefficients, using the coefficients of the previous level. Table 1 shows the schedule of a computation period for a 3 level DWPT. In this table the values in the column ‘input’ is the current inputs to DWPT structure. For instance the method for calculation of the $y_{1,1}(4)$ and $y_{2,3}(8)$ are presented as follows:

Based on Equation (1), in order to compute $y_{1,1}(4)$, the processor $P_0$ reads from $M_0$ the L inputs $y_{0,0}(0)$, $y_{0,2}(2)$, $y_{0,6}(6)$ and $y_{0,8}(8)$ in cp# 6, 7, 8 and 9. These inputs are passed through the high-pass filter of the processor $P_0$. The output coefficient of high-pass filter, $y_{1,1}(4)$, is written in memory $M_{1,1}$ in cp# 9. The output coefficient of low-pass filter, $y_{1,0}(4)$ is also calculated at the same time and is written in the memory $M_{1,0}$.

The following remarks about the updating of the memories are concluded from Table 1:

The content of each location in the memory $M_0$ is updated every $L/2$ clock cycles at CP# 0, 2, 4,…. The memory $M_0$ has $L$ registers, therefore this memory is updated every $2L$ clock cycles with the same pattern.

The content of each location in the memories $M_{1,1}$ and $M_{1,0}$ is updated every $L$ clock cycles at CP# 1, 5, 9,…. The memories $M_{1,1}$ and $M_{1,0}$ has $L$ registers, therefore this memory is updated every $2L$ clock cycles with the same pattern.

The content of each location in the memories $M_{2,3,1}$ and $M_{2,2,0}$ is updated every $L$ clock cycles at CP# 2, 6, 10,…. The memories $M_{2,3,1}$ and $M_{2,2,0}$ has $2L$ registers, therefore this memory is updated every $4L$ clock cycles with the same pattern.

### 3.4 data dependency and inhibition nodes

The DWPT architecture has to calculate any selected decomposition subtree, without any modification to the addresses generation circuits. The on-line applications of this architecture impose some limit to the computing time, which is dependent to the selected decomposition. This time limit is considered by bypassing the coefficients belonging to the terminal nodes from one memory to the following one in the pipeline [1]. Suppose that node (2, 1) inhibited to decomposing into nodes (3, 2) and (3, 3) (see Figure 1).

Coefficients, $y_{3,2}(0)$ and $y_{3,3}(0)$ have to replaced with coefficients $y_{3,2}(0)$ and $y_{3,3}(0)$. For instance coefficient $y_{3,3}(0)$ is replaced by one of these coefficients: $y_{3,2}(−12)$ or $y_{3,2}(−8)$ or $y_{3,2}(−4)$ or $y_{3,2}(0)$. All these coefficients are required to calculate the value of the coefficient, $y_{3,3}(0)$, in the uninhibited mode. These coefficients are stored in the memory $M_{2,3,1}$ and are read by processor $P_2$.

To determine which of inputs has to be allocated to each coefficient $y_{3,2}(0)$ and $y_{3,3}(0)$, the analysis dependencies is performed, and shown in Figure 4. These coefficients are selected with less latency, a multiplexer in the output of each filter selects output with less latency (see Figure 3).

When processor $P_2$ reads the $(L-1)$ th data from memory $M_{2,3,1}$, both the multiplier and the adder in the high-pass filter are inhibited and this $(L-1)$ th data is stored in the accumulator. The output of the accumulator is fed into the input ‘0’ of the multiplexer. When $P_2$ reads the $L$-th data stored in $M_{2,3,1}$, both the multiplier and the adder in the low-
Inhibition of a node in the level j inhibits all its descendents. For instance, inhibition of node (1,0) inhibits nodes (2,0) and (2,1). In Table 2, a part of bypass required for inhibition of node (1,0) is illustrated. In Table 2 ‘Ri’ is the inputs of processor P_i. ‘Yi’ is the data in the node (i,j) and ‘Pi’ is the processor target.

Table 2. Bypass required for inhibition of node (1,0)

<table>
<thead>
<tr>
<th>CPb</th>
<th>Y</th>
<th>R</th>
<th>P</th>
<th>Y</th>
<th>R</th>
<th>P</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>y_{0,0}^{0}</td>
<td>y_{0,0}^{1}</td>
<td>x_{0,0}^{0}</td>
<td>y_{0,0}^{0}</td>
<td>y_{0,0}^{1}</td>
<td>x_{0,0}^{0}</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>y_{0,0}^{0}</td>
<td>y_{0,0}^{1}</td>
<td>x_{0,0}^{0}</td>
<td>y_{0,0}^{0}</td>
<td>y_{0,0}^{1}</td>
<td>x_{0,0}^{0}</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>y_{0,1}^{0}</td>
<td>y_{0,1}^{1}</td>
<td>x_{0,1}^{0}</td>
<td>y_{0,1}^{0}</td>
<td>y_{0,1}^{1}</td>
<td>x_{0,1}^{0}</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>y_{0,1}^{0}</td>
<td>y_{0,1}^{1}</td>
<td>x_{0,1}^{0}</td>
<td>y_{0,1}^{0}</td>
<td>y_{0,1}^{1}</td>
<td>x_{0,1}^{0}</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>y_{0,2}^{0}</td>
<td>y_{0,2}^{1}</td>
<td>x_{0,2}^{0}</td>
<td>y_{0,2}^{0}</td>
<td>y_{0,2}^{1}</td>
<td>x_{0,2}^{0}</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>y_{0,2}^{0}</td>
<td>y_{0,2}^{1}</td>
<td>x_{0,2}^{0}</td>
<td>y_{0,2}^{0}</td>
<td>y_{0,2}^{1}</td>
<td>x_{0,2}^{0}</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>y_{0,3}^{0}</td>
<td>y_{0,3}^{1}</td>
<td>x_{0,3}^{0}</td>
<td>y_{0,3}^{0}</td>
<td>y_{0,3}^{1}</td>
<td>x_{0,3}^{0}</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>y_{0,3}^{0}</td>
<td>y_{0,3}^{1}</td>
<td>x_{0,3}^{0}</td>
<td>y_{0,3}^{0}</td>
<td>y_{0,3}^{1}</td>
<td>x_{0,3}^{0}</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>y_{0,4}^{0}</td>
<td>y_{0,4}^{1}</td>
<td>x_{0,4}^{0}</td>
<td>y_{0,4}^{0}</td>
<td>y_{0,4}^{1}</td>
<td>x_{0,4}^{0}</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>y_{0,4}^{0}</td>
<td>y_{0,4}^{1}</td>
<td>x_{0,4}^{0}</td>
<td>y_{0,4}^{0}</td>
<td>y_{0,4}^{1}</td>
<td>x_{0,4}^{0}</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>y_{0,5}^{0}</td>
<td>y_{0,5}^{1}</td>
<td>x_{0,5}^{0}</td>
<td>y_{0,5}^{0}</td>
<td>y_{0,5}^{1}</td>
<td>x_{0,5}^{0}</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>y_{0,5}^{0}</td>
<td>y_{0,5}^{1}</td>
<td>x_{0,5}^{0}</td>
<td>y_{0,5}^{0}</td>
<td>y_{0,5}^{1}</td>
<td>x_{0,5}^{0}</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>y_{0,6}^{0}</td>
<td>y_{0,6}^{1}</td>
<td>x_{0,6}^{0}</td>
<td>y_{0,6}^{0}</td>
<td>y_{0,6}^{1}</td>
<td>x_{0,6}^{0}</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>y_{0,6}^{0}</td>
<td>y_{0,6}^{1}</td>
<td>x_{0,6}^{0}</td>
<td>y_{0,6}^{0}</td>
<td>y_{0,6}^{1}</td>
<td>x_{0,6}^{0}</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>y_{0,7}^{0}</td>
<td>y_{0,7}^{1}</td>
<td>x_{0,7}^{0}</td>
<td>y_{0,7}^{0}</td>
<td>y_{0,7}^{1}</td>
<td>x_{0,7}^{0}</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>y_{0,7}^{0}</td>
<td>y_{0,7}^{1}</td>
<td>x_{0,7}^{0}</td>
<td>y_{0,7}^{0}</td>
<td>y_{0,7}^{1}</td>
<td>x_{0,7}^{0}</td>
<td></td>
</tr>
</tbody>
</table>

4. Inverse Wavelet Packet Transform architecture

Inverse Wavelet Packet Transform (IWPT) reconstructs the original signal by use the inverse filter bank corresponding to the one that show in Figure 1. Quadrature Mirror filters (QMF) is used for a perfect reconstruction. This makes possible to recognize computations, and to use the even (E) and odd (O) filters, consisting of the even and odd coefficients of the low-pass (H) and the high-pass (G) filters, respectively.

By using odd and even filters, IWPT architecture is similar to the DWPT architecture; however the computation stages are traversed in reverse order.

Processes that perform using the E and the O filters in IWPT architecture are similar to the processes that perform using the high-pass and the low-pass filters in DWPT architecture. Memories in the two architectures, DWPT and IWPT are also similar. Data dependency for the IWPT is illustrated in Figure 5.

5. Conclusions

In this paper based on the word-serial pipelined architecture and the parallel filter processing, using the high-pass and the low-pass filters, a new architecture for the wavelet packet transforms is introduced. Using two filters, one high-pass and one low-pass, concurrently, divides the latency of the memories to half.

In this architecture, the speed of the input sampling and the new coefficients product is twice as [1]. The delay of memories is half relative to the [1] and the volume of the memories are same.

This architecture is suitable for the direct and inverse WPT with any level decomposition and reconstruction. This high speed architecture is also suitable for the on-line applications.

6. References