PIPELINE ARRAY IMPLEMENTATION OF FIR FILTERS
Paraskevas Kalivas, Paul Bougas, Vassilis Vassilakis, Christos Meletis, Kiamal Pekmestzi

School of Electrical and Computer Engineering, National Technical University of Athens,
157 73 Zographou, Athens, Greece.
phone: +30 210 7722500, fax: +30 210 7722428, email: pekmes@microlab.ntua.gr
web: www.microlab.ntua.gr

ABSTRACT
A new pipeline array type parallel scheme for the implementation of FIR digital filters of low-latency is presented in this paper. Each cell of the array of the proposed scheme implements the computation of a one-bit FIR filter and is based on carry-save arithmetic. This structure leads to a low-latency implementation that is independent of the number of the filter taps. The proposed scheme is pipelined at the bit-level, requires less hardware and yields superior performance than other schemes that are based on discrete multipliers. Also, a merging technique is applied inside the one-bit FIR filter cell achieving systolicity at the bit-level.

1. INTRODUCTION
Among all signal processing algorithms, FIR digital filters are the most often implemented directly on chip. However the large number of filter tabs in practical applications requires significant amount of hardware. The continuous progress in VLSI technology eliminates this problem and allows designers to adopt parallel solutions, which achieve higher throughput rates. However, the need for hardware-efficient and pipelined implementations remains. Digital signal processing applications require low power consumption and high throughput rate. Therefore, hardware-optimized architectures play a significant role in the implementation of these systems.

In order to be suitable for the implementation of filters with large number of taps, the proposed schemes should be pipelined at the bit level. Also, their latency should be low and not proportional to the number of filter taps. The scheme mostly used for the implementation of FIR filters either in direct or in transposed form is based on a chain of multiplier-accumulators [1-3]. Such a scheme, converted to pipelined form has been presented in [4]. Its significant disadvantage is that latency is proportional to the number of filter tabs.

In [5] the use of the array architecture has been proposed for the implementation of FIR filters. The FIR filter computation has been moved inside the array cells at the bit-level. Each cell is a bit filter, namely it computes the convolution of input data bits with the same order bits of all filter coefficients. This architecture has the following advantages: systolic, with low latency and canonical structure. Also, in [5] the direct form has been chosen for the implementation of bit-level FIR filter and an internal pipelined accumulation tree has been used for the addition of the bit partial products.

Moreover, the technique of merging neighboring cells of the array is used, in order to obtain a systolic filter [5], [6]. The above implementation has the drawback of a large number of pipelining delays and latency proportional to \( \log_k k \), where \( k \) is the number of filter taps.

In this paper we propose a FIR filter scheme that is based on the architecture of carry-save multiplier array but we implement the bit-level FIR filters in transposed form. Furthermore, we suggest merging the neighbouring partial products at the bit-level, inside the cell. As a result, we achieve pipelining at the bit-level, significant reduction of pipelining delays and constant latency, independent of the number of filter taps. Moreover, in this scheme, merging can be applied at the level of the array cells in order to reduce broadcasting of data lines. Thus, the proposed scheme leads to modular systolic and hardware-efficient circuits.

The structure of this paper is as follows. In Section 2 we present the structure of the array. The detailed implementation of the array cells, namely the bit FIR filters is given in Section 3. In Section 4 we present the required modifications for the filter to perform computations with two’s complement numbers. In Section 5 we compare the proposed scheme with other schemes from the aspect of hardware complexity and latency.

2. THE PROPOSED FIR FILTER ARRAY
A K-tap FIR filter performs the computation given by the following equation

\[
y_n = \sum_{i=0}^{K-1} h_i x_{n-i}
\]

where \( x_n, n = 0, 1, 2, \ldots \) is the input sequence,
\( y_n, n = 0, 1, 2, \ldots \) the output sequence, and
\( h_i, n = 0, 1, 2, \ldots, K-1 \) are the filter coefficients. By writing the N-bit long data and the M-bit long coefficient words in binary form we get

\[
y_n = \sum_{i=0}^{K-1} \left( \sum_{n=0}^{M-1} h_{i,n} 2^n \right) \left( \sum_{n=0}^{N-1} x_{n} 2^n \right)
\]

This equation can be written as

\[
y_n = \sum_{m=0}^{K-1} \sum_{n=0}^{M-1} \sum_{i=0}^{N-1} h_{i,m} x_{n+m} = \sum_{m=0}^{K-1} \sum_{n=0}^{M-1} \sum_{i=0}^{N-1} h_{i,m} x_{n+m} p_{m,n}^b
\]

The above expression shows that a FIR filter can be realized by a scheme that has the structure of an array multiplier. The quantity \( p_{m,n}^b \) is an FIR bit-filter, namely a filter where both
coefficients and data have 1-bit length. A pipelined 4x3 carry-save array that implements the equation (3) is adopted, with each cell performing the operation of a bit-filter. The above array is shown in Fig. 1.

Each j-th diagonal coefficient line in Fig. 1 carries the K bits $h_j^0, h_j^1, \ldots, h_j^{K-1}$ of coefficients. Each of them belongs to a different filter coefficient. All the coefficient bits $h_j^i$ enter the circuit synchronously. We assume that coefficients remain constant during the operation of the filter. In case of coefficients, changing on the fly, delay elements should be placed on coefficient lines. The horizontal data lines correspond to the bits of $x_n$ that enter the circuit in bit-skew form.

Each bit-filter cell adds K partial products according to equation (3). Also, it adds the sum from the above vertical cell and K carries from the above right diagonal cell and outputs a sum and the carries for the lower cells. The least significant part of the output of the array is in bit-skew form and the most significant part in an extended carry-save form. We use the term 'extended carry-save' because the sum is 1-bit but the carry is K-bit long, as it will soon become clear. A row of cells at the bottom of the array converts the most significant part from extended carry-save into conventional carry-save form. This row is extended leftwards by $\log_2 K$ cells in order to prevent the result from overflowing. Apparently, the extra cells are simplified as we move leftwards and therefore are shown in Fig. 1 with smaller rectangular boxes. If the final result must be obtained in binary form, a pipelined vector merged adder is required.

3. III. THE IMPLEMENTATION OF THE CELLS OF THE FIR FILTER ARRAY

Each array cell computes the bit-filter $p_{n,m}^{w}=\sum_{j=0}^{K-1} h_j^w x_{n-j}^w$ given by equation (4).

As it is shown in Fig. 2a, the bit-filter is implemented in transposed form. The carry output of each Full-Adder is connected to the carry input of the corresponding Full-Adder in the lower left diagonal array cell. For example, the carry output of the first Full-Adder is connected to the carry input of the first Full-Adder of the next diagonal cell, the second to the second etc. An extra Full-Adder at the bottom of the circuit adds the sum output from the above array cell. Adding the sum at this position allows all the array cells of the same column to operate with one clock cycle difference. This arrangement yields low latency, independent of the array rows number.

Figure 1: The structure of the proposed FIR filter scheme with K filter taps and bit-length $w=4$.

Figure 2: The detailed implementation of the array cell for FIR filter with 8 filter taps (a) unmerged (b) merged.
clock cycle corresponds to the last addition stage. A disadvantage of the circuit in Fig. 2a is the lack of systolicity, because the line of \( x_i \) is broadcasted to all gated Full Adders. Systolicity at the cell level is important due to the large number of taps in the FIR filters that are used in practical applications. We use the merging technique inside the cell [5][6], in order to transform the circuit into systolic form. According to this technique, pairs of neighboring gated Full Adders are merged and the retiming moves the intermediate delay to the line of \( x_i \) without affecting the carry lines. Thus, we have pairs of Full-Adders that form the merged sub-cells of the bit-filter. The first sub-cell especially requires only one Full-Adder since it has no sum input to add. The delays in the data line shown in Fig. 2b are not repeated at each array cell, because the input and output of each delay are broadcasted to all the cells of a row. They exist only once at the beginning of each row of the array.

In order to keep the combinational delay of the circuit equal to the delay of one gated Full-Adder, a delay element is inserted between the merged Full-Adders in each sub-cell. This way we insert an internal pipelining in the array cell, which changes the timing of the bit-filter. This timing is shown with the numbers next to each output in Fig. 2b. This change does not affect the operation of the whole circuit because the internal pipelining is applied to all cells. Thus the carry outputs of each array cell are connected to the carry inputs of the diagonal cell whose timing has been affected in the same way.

The comparison of the timing of the circuits in Fig. 2a and 2b reveals that the merging technique reduces the pipeline stages and consequently the timing gap between the beginning and the end of the bit-filter operation at the expense of increased by one clock cycle latency. Particularly, the time gap of a bit-filter operation is \( \frac{K}{2} + 1 \) clock cycles for the merged sub-cell and \( K \) clock cycles for the unmerged.

The sum and carry outputs produced by the bottom row of the array have to be added by an extra row of cells, in order to produce the result in carry-save form. Each of these sum cells has to add the sum and carry outputs of the above bit-filter, and the carry outputs of the right sum cell as can be seen in figure 1. Two connected sum cells are shown in Fig. 3. Each sum cell consists of two columns of Full-Adders. The first column includes \( \frac{K}{2} \) Full-Adders and adds the carry-outputs of above the array-cell. The sum \( s_i \) of the above array-cell is added at the bottom of this column to keep the overall latency of the circuit low. The second column consists of \( \frac{K}{2} - 2 \) Full-Adders and adds the carry outputs from the neighbouring right sum cell. A Full-Adder at the bottom of the cell adds the sums of these columns and produces the result in carry-save form.

It is apparent that another effect of the merging applied to the array-cells is the reduction of the hardware complexity of the sum cells since the reduction of the timing gap reduces the pipelining stages in the sum cell. The sum line adds only two clock cycles to the latency. The total latency of the circuit is constant and equal to four clock cycles.

Figure 3: The detailed implementation of two connected sum cell for an 8th order FIR filter.

The latency of the proposed scheme, namely the timing gap between the input of the LSB of \( x_n \) and the production of the most significant part of the result is

\[
L = w + L_{\text{array-cell}} + L_{\text{sum-cell}}
\]  

where \( w \) is the data word length of \( x_n \), \( L_{\text{array-cell}} \) the latency of the array-cell, which is equal to two clock cycles, and \( L_{\text{sum-cell}} \) the latency of the sum-cell which is equal to two clock cycles. Thus the latency is \( L = w + 4 \) clock cycles. The term \( w \) in equation (5) is due to the bit-skew form of \( x_n \). The same term exists in the case of implementing the filter using discrete array multipliers and corresponds to the latency of the pipelined implementation of these multipliers.

The above scheme can be implemented by rearranging the inputs of the array, so that the filter coefficients \( h_j \) enter the array from the top and the data \( x_n \) from the right. In this case the diagonal lines of \( x_n \) require an extra delay element between the cells. These increases the hardware complexity since \( w \cdot m \) extra delays are required, where \( w \) and \( m \) are the bit-lengths of \( x_n \) and \( h_j \) respectively. When \( x_n \) enters array from the right a triangle of \( \frac{w^2}{2} \) delays is required for the conversion of \( x_n \) from parallel into bit-skew form. However, when the filter coefficients with bit-length significantly shorter than that of \( x_n \) the above rearrangement reduces the hardware and the latency, which is \( L = m + 4 \) in this case.
4. COMPARISON OF THE PROPOSED SCHEME WITH OTHER SCHEMES

The latencies of the schemes presented in [4], [5] and of the proposed scheme are given in Table I. In this table we assume as latency the number of the clock cycles from the input of the first bit of the data $x_n$ to the output of the most significant part of the results. Only the proposed scheme has latency independent of the number of filter taps.

Table I: Latency comparison of the proposed schemes.

<table>
<thead>
<tr>
<th>Filter scheme</th>
<th>Latency in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presented in [4]</td>
<td>$w + (k - 1) + 2$</td>
</tr>
<tr>
<td>Presented in [5]</td>
<td>$w + \log_2 k + 3$</td>
</tr>
<tr>
<td>Proposed scheme</td>
<td>$w + 4$</td>
</tr>
</tbody>
</table>

The hardware complexities of the three schemes are given in Table II. We assumed that the first scheme is implemented with carry-save multipliers pipelined at the bit-level and that the intermediate results are not truncated. For computational simplicity we assumed that input samples and filter coefficients have the same bit length $w$ and that the $\log_2 k$ extra sum cells used for avoiding the accumulation overflow in the proposed scheme are not simplified. The AND gates are not included in this table because their number is the same in both schemes. In the proposed scheme, the hardware complexity of the array cells includes the delays in the line of $x$ that are common to all the array cells that belong to the same row.

Table II: Hardware complexity of the proposed scheme.

<table>
<thead>
<tr>
<th>Filter scheme</th>
<th>Full-Adders</th>
<th>Delays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presented in [4]</td>
<td>$kw^2 + 2kw' + 2kw$</td>
<td>$2kw^2 + 5kw' + 6kw$</td>
</tr>
<tr>
<td>Presented in [5]</td>
<td>$kw^2 + (k - 1)w'$</td>
<td>$\frac{5k + 1}{2}w^2 + \frac{5k}{2}w'$</td>
</tr>
<tr>
<td>Proposed scheme</td>
<td>$kw^2 + (k - 1)w'$</td>
<td>$2kw^2 + \frac{5k}{2}w' + \frac{k + 1}{2}w$</td>
</tr>
</tbody>
</table>

$k$: Number of filter taps

$w = w + \log_2 k$: The digit-length of the most significant part of the output.

The comparison between the first and the proposed scheme reveals that constant latency in the proposed scheme is achieved without increasing the total hardware complexity. On the contrary, we have some hardware saving due to the reduced hardware complexity of the row of sum-cells. Actually, the sum cells hardware complexity is even lower because the $\log_2 k$ extra sum cells have less hardware than normal. The corresponding hardware reduction becomes more significant as the number of filter taps increases.

In the scheme presented in [5] a triangle of array element is proposed for the summation of the array output. Obviously, the summation can be implemented more efficiently with a row that consists of the sum cells presented in this paper. In order the comparison to be fair the hardware complexity of the second scheme in TABLE II has been computed according to this assumption. The comparison shows that the array-cell of the proposed scheme requires less delay elements than the array-cell of the second scheme.

5. CONCLUSION

A new FIR filter scheme suitable for unsigned and signed computations is presented in this paper. It is based on the structure of the carry-save array multiplier and is pipelined at the bit-level. Every cell of the array is a bit-filter in transposed form. This architecture achieves latency independent of the number of filter taps and requires less hardware than other schemes based on discrete multipliers. Applying the merging technique inside each cell leads to systolicity at the cell-level. The systolicity at the array-level is not critical in most DSP applications because the bit-length of the filter coefficients and input data are relatively small compared to the number of taps. However, this type of systolicity can be achieved in the proposed architecture by applying the merging technique at the array-level, namely by merging whole array cells.

REFERENCES